

# **AllChange Administrator Manual**

---



# AllChange Administrator Manual

Version 10.0, Sep 2014



Email: [support@intasoft.net](mailto: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

---

<b>What's New in AllChange 10.0</b> .....	<b>1</b>
Keyword Substitution Improvements .....	1
Baseline Details have Arbitrary Fields .....	1
Web Browser Access Updates .....	1
<b>Introduction</b> .....	<b>2</b>
Introduction .....	2
<b>Getting Started</b> .....	<b>3</b>
Installing AllChange .....	3
Workstation Installation .....	3
First Steps .....	4
The AllChange Configuration Editor — ACCONFIG .....	4
About The AllChange Configuration Editor — acconfig .....	4
Invoking acconfig .....	4
The Main acconfig Window .....	4
The acconfig Main Menu .....	5
acconfig Options .....	5
Saving Changes in acconfig .....	7
Editing in acconfig .....	7
The AllChange Text Editor .....	8
AllChange Projects .....	17
Licensing Users .....	18
Allocating Users to Roles .....	19
Defining Workspaces .....	20
<b>Setting up your Environment</b> .....	<b>22</b>
About Setting up your Environment .....	22
Defining the AllChange Environment .....	22
SQL Server .....	23
Microsoft SQL Server 2008 .....	23
Microsoft SQL Server 2008 Express Edition (SQL Express) .....	23
Installation .....	24
Creating an AllChange Project Database .....	25
Accessing an AllChange Project Database .....	25
SQL Server Permissions .....	28
Accessing the Database Outside of AllChange .....	28
Relationship Between SQL Server and AllChange Server .....	28
Modes of Operation .....	29
About Modes of Operation .....	29

---

Direct .....	30
Client/Server .....	30
Security and Access Issues .....	40
About Security and Access Issues .....	40
Directory Permissions .....	40
AllChange Login .....	42
Integrations .....	43
About Integrations .....	43
Integrated FTP support .....	43
Import from Email .....	46
AllChange Web Browser Access .....	48
Web Development Support .....	59
Graphical Statistics .....	59
Integration with Microsoft Project .....	61
About Integration with Mail .....	62
Development Tool Integrations .....	64
Integration with Eclipse .....	66
Integration with Araxis Merge .....	67
Integration with Windows Explorer .....	67
Integration with Microsoft Word/Excel .....	68
Integration with Dreamweaver .....	71
Integration with HP TestDirector .....	72
Integration with Telelogic Rhapsody .....	75
Integration with Telelogic DOORS .....	75
File Comparison Tools .....	77
Integration with SolidWorks .....	77
Windows Registry Settings .....	78
Licensing .....	80
<b>AllChange Projects .....</b>	<b>82</b>
About AllChange Projects .....	82
AC4AC Project .....	83
Opening a Project .....	83
Creating and Updating Projects .....	84
About Creating and Updating Projects .....	84
Initialising the Database .....	91
About Using AllChange Projects .....	91
Project Templates .....	92
Project Templates .....	92

---

Intasoft Standard Configuration .....	92
ControlledSource Cycle Driven Change Process.....	96
ReviewableDoc Cycle Driven Change Process.....	97
Derived Cycle Driven Change Process.....	98
Websrc Cycle Driven Change Process.....	99
ITAsset Cycle Driven Change Process.....	100
SolidWorks Document Cycle Driven Change Process.....	101
Change Process with no Life-Cycle.....	102
SCR Change Management Process.....	104
Generic Change Management Process.....	105
Work Order Change Management Process.....	107
ITIL Configuration.....	111
Normal Change Process.....	127
Emergency Change Process.....	129
Standard Change Process.....	130
IntaChange Integrate Template.....	137
<b>Parts/Files and Database Relationships.....</b>	<b>141</b>
About Parts/Files and Database Relationships.....	141
The Database.....	141
Parts, Files and the Database.....	141
Variable Locations.....	143
<b>Regular Administrative Tasks.....</b>	<b>144</b>
About Regular Administrative Tasks.....	144
Resolving Discrepancies.....	144
Backing up.....	145
Backing Up and Restoring the Database in acconfig.....	146
<b>The AllChange Configuration Facilities.....</b>	<b>147</b>
About The AllChange Configuration Facilities.....	147
Defining Access Control.....	147
About Defining Access Control.....	147
User Registration.....	147
User Groups.....	149
Lockout Times.....	150
Role Definitions.....	152
Command Access.....	154
Defining your Configuration Management Plan.....	159
About Defining your Configuration Management Plan.....	159
Configuration Options.....	159

---

Classes.....	178
Life-cycles.....	184
Function Definition Files.....	199
Command Definitions.....	201
Vote Definitions.....	202
The Startup File.....	206
Branch Names.....	207
Report Formats.....	207
Scheduled Jobs.....	209
Defining Workspaces and Pools.....	213
About Defining Workspaces and Pools.....	213
Workspaces.....	214
Pools.....	217
Web Mapping.....	219
Configuring the GUI.....	219
About Configuring the GUI.....	219
Field Name Definitions.....	219
Field Tabs.....	226
Menu Items.....	227
Menu Item/ Toolbar Availability.....	229
Menu Actions.....	231
Browsers.....	232
ACCEL Condition Editor.....	236
Nomenclature Mapping.....	238
Project Dictionary.....	241
Intercepting Viewer Buttons.....	241
Register Definition Files.....	242
Creating Monitor Events.....	242
Administrator Facilities.....	243
ace Facilities for Administrators.....	243
Debugging.....	245
URL Protocol.....	246
<b>Creating and Modifying Report Formats.....</b>	<b>248</b>
About Creating and Modifying Report Formats.....	248
Adding New Formats to ace.....	248
Columnar Reports in Excel/Word.....	249
Report Format Syntax.....	249
About Report Format Syntax.....	249

---

Control Section .....	249
Break Section .....	251
Display Section .....	252
Image Section .....	252
Databases .....	256
Sample Formats .....	261
Examples .....	262
Integration with Excel Charts .....	263
<b>Upgrading to a New Version of AllChange .....</b>	<b>264</b>
About Upgrading to a New Version of AllChange .....	264
The Upgrade Tool – ACUPGRAD .....	265
About The Upgrade Tool – acupgrad .....	265
The Upgrade Process .....	265
The Upgrade Wizard .....	267
Upgrade Sessions .....	273
Performing the Upgrade Stages .....	274
Options .....	278
Going Live .....	279
Creating a Copy of an AllChange Project .....	280
<b>ACCEL — AllChange Command Evaluation Language .....</b>	<b>282</b>
About ACCEL — AllChange Command Evaluation Language .....	282
Notation .....	282
ACCEL Expressions .....	283
ACCEL Statements .....	283
ACCEL Conditions .....	289
ACCEL Field References .....	291
About Database Fields .....	291
About ACCEL Field References .....	293
Field References for Parts Database .....	294
Field References for Check-outs Database .....	297
Field References for Instances Database .....	297
Field References for Baselines Database .....	298
Field References for Monitors Database .....	299
Field References for CRs Database .....	300
Field References for Status Logs Database .....	302
Field References for Items Affected Database .....	303
Field References for Votes Database .....	304
Field References for Pools Database .....	304

---

Field References for Classes Database .....	305
Field References for Workspaces Database .....	305
Field References for Roles Database .....	306
Field References for Cycles Database .....	306
Field References for Command Definitions Database .....	307
Field References for Fieldnames Database .....	307
Field References for Field Tabs Database .....	307
Field References for Rolemap Database .....	307
Field References for Configuration Options Database .....	308
Field References for Includes Database .....	308
Field References for Report Formats Database .....	308
Field References for User Registrations Database .....	308
Field References for Groups Database .....	309
Field References for Nomenclature Mapping Database .....	309
Field References for Branches Database .....	309
Field References for Part Permissions Database .....	309
Field References for the Vote Definitions Database .....	311
<b>ACCEL Functions .....</b>	<b>312</b>
About ACCEL Built In Functions .....	312
List of ACCEL Built In Functions by Category .....	312
A–Z list of ACCEL Built In Functions .....	315
User-defined Functions .....	415
"Dangerous" Functions .....	416
<b>ACCEL Variables .....</b>	<b>417</b>
User-defined Variables .....	417
Built In Variables .....	418
<b>Visual ACCEL .....</b>	<b>426</b>
About Visual ACCEL .....	426
Callbacks .....	433
Notes .....	433
Wizard Dialogs .....	434
<b>AllChange Project Settings Editor .....</b>	<b>435</b>
About AllChange Project Settings Editor .....	435
Modifying the Registry .....	436
Importing "ini" Files .....	436
<b>Appendix A - Command Definitions in Depth .....</b>	<b>437</b>
Command Definitions in Depth .....	437
addbaselinevote .....	437

---

addcrvote .....	438
addpartvote .....	438
alter .....	438
alterbaseline .....	438
alterbaselinevote .....	438
altercr .....	439
altercrvote .....	439
alterinstance .....	439
alterissue .....	439
alterpartvote .....	439
alterversion .....	439
assigncr .....	440
attachws .....	440
baseline .....	440
build .....	440
command .....	440
copy .....	440
cwp .....	441
delete .....	441
deletebaseline .....	441
deletebaselinevote .....	441
deletecr .....	441
deleteissue .....	441
deletecrvote .....	442
deleteinstance .....	442
deletemonitor .....	442
deletepartvote .....	442
deleteversion .....	442
edit .....	442
editcrtext .....	442
eval .....	443
find .....	443
getcrtext .....	443
ignoreprotecttext, ignoreprotecttextbaseline, ignoreprotecttextpart .....	443
interpret .....	443
issue .....	444
list .....	444
monitor .....	444

---

monitor_action.....	444
newcr.....	445
newinstance.....	445
newversion.....	445
promote.....	445
putcrtext.....	445
quit.....	446
readbaseline, readbaselinevote, readcr, readcrvote, readinstance, readmonitor, readpart, read-... partvote, readbaselinestatuslog, readcrstatuslog, readpartstatuslog.....	446
release.....	446
rename.....	447
renamebaseline.....	447
report.....	447
return.....	447
role.....	448
set.....	448
status.....	448
statusbaseline.....	448
statuscr.....	448
use.....	449
voteblocked.....	449
votedecision.....	449
voteinitiated.....	449
votepassnext.....	449
<b>Appendix B - Integration with Version Control Tools.....</b>	<b>450</b>
Appendix B — Integration with Version Control Tools.....	450
The VC System.....	450
VC Keywords.....	450
<b>Appendix C - System Configuration.....</b>	<b>451</b>
Appendix C — System Configuration.....	451
System Files.....	452
<b>Appendix D - Windows Operating System Information.....</b>	<b>468</b>
Appendix D — Windows Operating System Information.....	468
<b>Glossary.....</b>	<b>469</b>
<b>Index.....</b>	<b>474</b>

## What's New in AllChange 10.0

AllChange Version 10.0 contains several new features, as well as enhancements to existing features. Also, this release contains bug fixes. New and updated features are listed below, with links to the relevant section of the documentation where appropriate. See the UPGRADE.ME file for information on those not listed here.

[Keyword Substitution Improvement](#)

[Baseline Details have Arbitrary Fields](#)

[AllChange Web Browser Access Updates](#)

### Keyword Substitution Improvements

A couple of changes have been made to keyword substitutions

1. The limit on the length of a line which has keyword substitutions has been doubled to 508 characters.
2. Configuration options have been introduced to aid in preventing unwanted keyword substitutions on binary files. They attempt to detect when a workfile *looks like* a binary file, yet keyword substitutions are enabled by default and the VC file has not been *explicitly* marked as either enabled or disabled for keyword substitutions: see [Configuration Options](#) for details

### Baseline Details have Arbitrary Fields

Baseline details now have up to 40 arbitrary fields which may be defined. This may be useful for storing information about a detail specific to a particular baseline. For example with a Bill Of Materials this might be used for a Quantity These are defined in the normal way in the [Field Name Definitions](#) and may be viewed and updated from a new [Baseline Detail Viewer](#).

### Web Browser Access Updates

AllChange Web Browser Access Windows Authentication has been updated so that it supports Windows 2008 R2 or later domain. We have fixed this by introducing a different component to do the authentication. We have also used this opportunity to update all the supplied components, which were getting quite old, with the latest versions.

The components supplied are now:

- JRE 7 Update 55 (Java Runtime Environment)
- Apache Tomcat 6.0.41 (Web server)
- Commons FileUpload 1.3.1 (File upload/download transfer)
- Waffle 1.5 (Windows Authentication Filter --- replaces old jcifs)

See [AllChange Web Browser Access](#)

# Introduction

## Introduction

**AllChange** is an extremely powerful and flexible system. The **AllChange** Administrator is an individual, or group of individuals, with a number of tasks to perform to support end users and the smooth operation of the system.

This manual describes these tasks. It is organised as follows:

- [Getting Started](#) describes the basic tasks required to get you started using **AllChange**.
- [Setting up your Environment](#) describes how to set your **AllChange** operating environment.
- [AllChange Projects](#) describes how to create and use **AllChange**'s concept of *projects*.
- [Parts/ Files and Database Relationships](#) describes the relationships between parts, files and the database.
- [Regular Administrative Tasks](#) describes the regular administrative tasks that need to be performed to ensure that **AllChange** runs smoothly and efficiently.
- [The AllChange Configuration Facilities](#) describes how to configure **AllChange** projects to your specific configuration management requirements.
- [Creating and Modifying Report Formats](#) describes the layout of **AllChange** report format files. This knowledge will be required in order to tailor existing reports and create new ones.
- [Upgrading to a New Version of AllChange](#) describes the steps involved in upgrading from one release of **AllChange** to another.
- [Database Fields](#) describes the fields held in each of the **AllChange** databases in detail. These, together with [Classes](#), [Life-cycles](#), [Command Definitions](#), [Workspaces](#), [Pools](#) and [Role Definitions](#), will be required when writing reports and tailoring **AllChange**'s configuration to site requirements.
- [ACCEL — AllChange Command Evaluation Language](#) describes the **AllChange** Command Evaluation (or programming) Language (ACCEL) which is fundamental to tailoring the system.
- [Command Definitions in Depth](#) describes the requirements of the **AllChange** `commands.ac` file, which will be of use if altering the functionality of the system.
- [Integration with Version Control Tools](#) describes integration between **AllChange** and the version control tools which supply the low level version control and system building facilities.
- [System Configuration](#) summarises the use of system files by **AllChange**.
- Operating System Specifics describes operating system specific issues.

The **AllChange** Administrator will, at some time, need to read all of these sections.

In view of the tailorability of the system, the Administrator (or whoever configures the system) will also need to think about how to convey the site's procedures and intended use of the system to end users. This may entail writing site-specific documentation and instructions.

# Getting Started

## Installing AllChange

To install **AllChange** you should run the program downloaded from the Intasoft website.

Installing **AllChange** involves two stages:

1. First a **Full** installation should be performed. This will copy the required **AllChange** program and configuration files (including the installation program itself) onto a hard disk (usually a network directory if **AllChange** is to be used by more than one person).
2. Then a **Workstation** install should be performed for each workstation that is to use **AllChange**. This will install some files onto the local workstation where this is necessary, create registry entries and set up the required environment for **AllChange** to run.

When the installation program is run you will be presented with various options which allow you to perform Full and Workstation installs. You should perform a **Full** install once initially. *Note that the **Full** install option will also perform a workstation install for the workstation performing the full install.*

The Full install will request that you enter your **AllChange** Serial number and License code, these will have been provided to you by Intasoft.

After the full install and when you are ready for other users/ workstations to use **AllChange** a **Workstation Install** should be performed for each workstation. In order to perform a workstation install you should run the `setup.exe` in the `setup` subdirectory of where you installed **AllChange**.

For more details of the installation see the **Help** for each screen of the installation. See also [Installation](#) in the Getting Started Guide and [SQL Server Installation](#).

Additional details may also be found in [Defining the AllChange Environment](#), [Setting up client/server — Summary](#), [Development Tool Integrations](#), [Integration with Windows Explorer](#) and [Integration with Microsoft Word](#).

**AllChange** is an extremely flexible system. Much of the flexibility is achieved as a result of the fact that its behaviour at many stages can be tailored to suit each individual site's requirements and wishes. In principle users need not be aware of the distinction between what is coded into the system itself and what has been grafted on top of it at their local site. However, to achieve a system which conforms to the site's desires the **AllChange** System Administrator must be prepared to come to terms with the ideas on which the system is founded and the details of how to set up and make changes in the system.

Having installed **AllChange** it must be correctly set up before *anybody* can use it.

As the **AllChange** Administrator you will need to follow the steps below before proceeding further:

1. License all users
2. Create an **AllChange**Project
3. Allocate roles to users
4. Define workspaces for users
5. Install the software on each workstation

These steps are discussed in [Getting Started](#).

The above steps are the minimum configuration requirements for using **AllChange**. However, this setup may not suit your particular requirements and you may wish to further tailor the many configurable aspects of **AllChange**. These facilities are detailed in the remaining chapters of this manual.

### Workstation Installation

Having installed **AllChange** from the media on which it was supplied onto your network, you will also need to perform a workstation installation for each user who is to have access to **AllChange**.

The workstation installation will simply set up some shortcuts and make some registry entries as well as ensuring that you have the correct versions of various libraries etc.

In order to perform a workstation install you will need rights to write to the HKEY\_LOCAL\_MACHINE section of the registry - this probably requires that you have administrator rights for the workstation.

Depending on your workstation settings this may require you to log in as the administrator and perform the workstation installation without selecting any of the integrations. Having done this you should then log in as your normal user, and if you wish to use any of the integrations (MS Office, Explorer, MCSCCI) you should perform the workstation installation again.

In order to perform a workstation install you should navigate in Windows Explorer to the **AllChange** installation directory on the network, find the `setup` subdirectory and run the `setup.exe` program.

Note that users will require direct access to the `muen.ac` and `users.ac` files in the **AllChange** installation directory during installation

Follow the instructions on the screen selecting the options required for the user.

## First Steps

**AllChange** is an extremely versatile and configurable tool allowing you to implement whatever configuration management processes are required for your particular site/ project.

Having installed **AllChange** you will need to perform the basic set up operations a listed below in order to get started using **AllChange** with the supplied *out-of-the-box* configuration management processes.

You will need to:

1. [Create an AllChange Project](#)
2. [License all users](#)
3. [Allocate roles to users](#)
4. [Define workspaces for users](#)
5. [Install the software on each workstation](#)

Once you have familiarised yourself with **AllChange** you may then wish to tailor the configuration to better suite your particular requirements. The rest of this manual is dedicated to providing you with the details of how this may be performed.

## The AllChange Configuration Editor --- ACCONFIG

### About The AllChange Configuration Editor — ACCONFIG

ACCONFIG is an administrative tool for use in setting up **AllChange** as required by *users*. It is intended for use by the **AllChange** Administrator and should not be made available to normal **AllChange** users.

[The AllChange Configuration Facilities](#) describes in detail all the various different configuration options and facilities. ACCONFIG saves the configuration information in various files which reside in the project and system directories. These files will be seen when browsing the project directory using, for example, Windows Explorer. These files will have the suffices `rep`, `acr`, `ac`, `ini`, `bak`, `sum`.

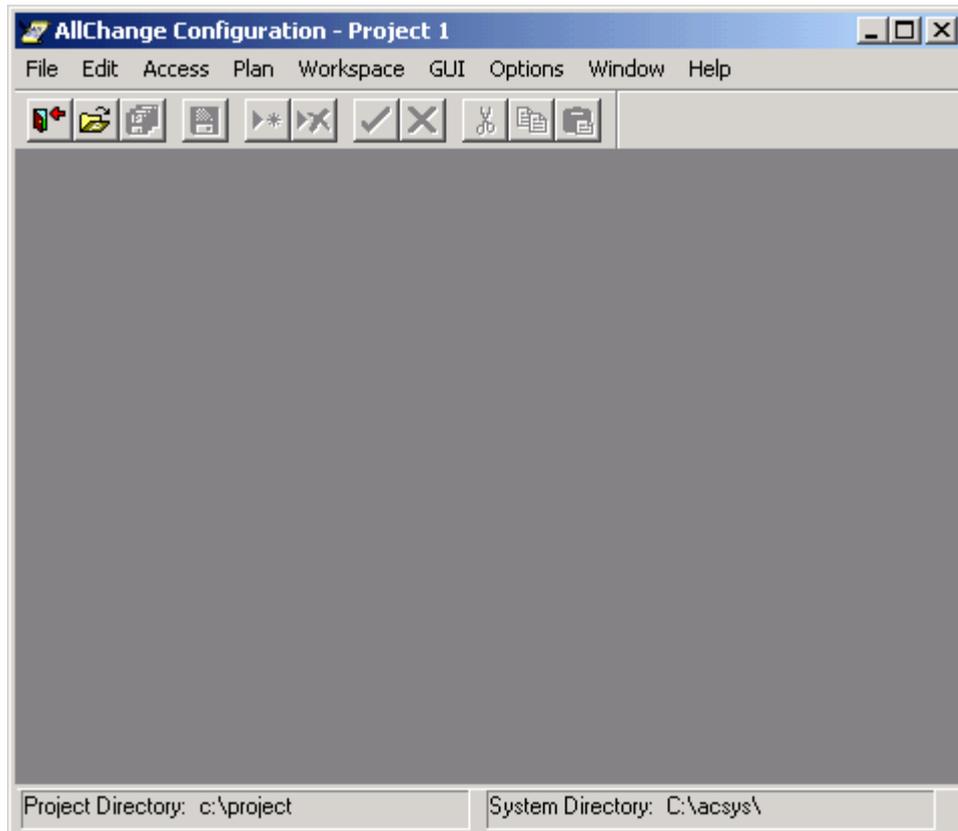
### Invoking ACCONFIG

A shortcut to ACCONFIG will be created on any workstation on which an Administrator workstation install has been performed. This will invoke ACCONFIG for the primary/ default **AllChange** project.

ACCONFIG allows **AllChange** projects to be configured as required. Environment variables may be set from the command line by using `-Evar=value`, e.g. the **AllChange** project may be set using `-EAC-PROJECT=...`, see [Setting up your Environment](#).

### The Main ACCONFIG Window

ACCONFIG provides a multi-windowed GUI interface to the **AllChange** configuration and set up facilities.



On startup ACCONFIG will automatically open the *current AllChange* project as determined by environment variables, the *registry* and the project definitions that exist. The project directory currently open is shown in the title bar; it is also shown in the status bar at the bottom of the window together with the *AllChangesys-* system directory .

### The ACCONFIG Main Menu

The Main Menu:

- provides access to different *AllChange* projects (**File**),
- allows items to be modified (**Edit**),
- provides access to the different configuration facilities (**Access, Plan, Workspace, GUI**),
- allows ACCONFIG itself to be configured, (**Options**)
- and provides access to the standard **Window** and **Help** facilities.

The **Access** menu allows you to define the access control requirements of a project.

The **Plan** menu gives access to the facilities for defining/ implementing a project's *configuration management plan*.

The **Workspace** menu provides access to the workspace management facilities.

The **GUI** menu provides access to various facilities for tailoring the ACE interface.

### ACCONFIG Options

The **Options** menu provides access to various options which modify the behaviour of ACCONFIG.

#### Confirm on delete:

If selected requires confirmation to be given for the deletion of items

#### Check ACCEL Syntax:

If selected causes ACCEL code modifications to be checked for syntactic correctness. This option is ignored if **Don't Wait for Editor** is selected.

**Don't Wait for Editor:**

If selected causes ACCONFIG to allow asynchronous editing of text files (e.g. function definition files and command access file) i.e. not to wait for the external text editor to close before allow the user to continue working in ACCONFIG.

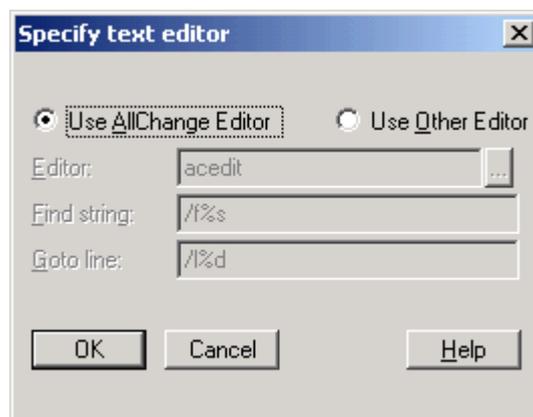
There are a few points to consider before selecting this option:

- The file will not be checked for syntax errors. Normally, ACCONFIG will (if the **Check ACCEL Syntax** option is enabled) check for syntax errors in the edited file, and offer to re-edit in the event of an error.
- If the file did not already exist in the project directory, then ACCONFIG normally copies the edited file to the project directory only if it has changed. If editing asynchronously then the file will be copied to the project directory unconditionally.
- ACCONFIG will usually allow files to be edited even if they are read-only. It then makes the file writable for saving, and then makes the file read-only again. Editing asynchronously means that ACCONFIG can not do this. If the default editor is the **AllChange** Editor (ACEDIT) then this will be passed a command-line option (-w) to instruct it to allow changes to be made to a read-only file, leaving it read-only after saving. Note that if the file is writable before being edited then it is left writable. If an alternative editor is used, then the file is made (and left) writable unconditionally.
- The cycles file may not be edited asynchronously, as ACCONFIG re-reads its contents in order to display changes to cycles and statuses in the cycle editor window.
- It should be noted that when editing files asynchronously ACCONFIG has no knowledge of whether a file is still open in an editor, and it is therefore possible to instruct ACCONFIG to open the file a second time for editing.

This is not recommended, as it can be easy to overwrite one set of changes with the other. It is perfectly safe to quit while editing a file in this way ACCONFIG

**Editor:**

Allows you to specify which editor you would like to use for modifying ACCEL code. An editor is supplied called **ACEdit** and this will be used by default, see [The AllChange Editor](#) for details of the editor.



**Editor** should specify the path to the program you wish to use as your editor. This must be a text editor.

**Find String** should specify the command line argument to the editor which allows a string to be searched for on startup. Use %s to indicate where the search string should be substituted by ACCONFIG when using this facility.

**Goto Line** should specify the command line argument to the editor which allows a line number to be specified as the starting line to show for a file on startup. Use %d to indicate where the line number should be substituted by ACCONFIG when using this facility.

**Toolbar:**

Allows the toolbar configuration to be modified

**Saving Changes in ACCONFIG**

Changes to the current upfront window may be saved using **File | Save Window** or **Edit | Save Window**.

All changes to the open project may be saved using **File | Save Project**.

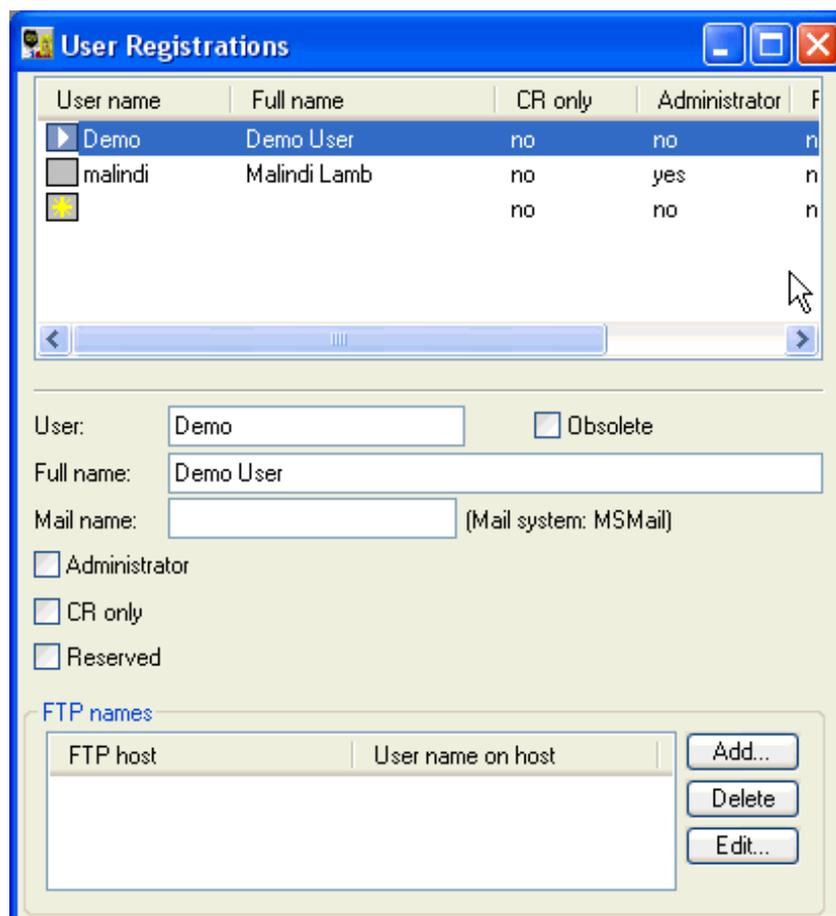
Any time an attempt is made to close a project or quit ACCONFIG you will be warned of any unsaved changes and given the option to save or discard the changes.

**Editing in ACCONFIG**

Modifications to all the configurable facilities within **AllChange** are made by opening the appropriate window accessible from the main menu.

Modifications may then be made to any item in place in the window — these changes will not be registered until either new information is displayed in that window or **Edit | Accept Changes** is selected. Until changes to a window have been registered they may be discarded using the **Edit | Discard Changes**.

Many of the windows have the form of a list of items in the top half of the window and the details of the currently selected item in the bottom half. The **User Registration** window is an example of this and is shown below.



The currently selected item and a new blank item in the list part will be indicated by an icon and shown below:

Icon	Description
	currently selected

	item
	new blank item

If it is desired to create a new item in a window this may be achieved using **Edit | New Item**, alternatively a blank entry is always available. Once a new item has been selected in the list part of a window then the details of the new item may then be entered in the item part of the window.

Similarly to delete an item, simply select it in the list part of the window and select **Edit | Delete Item**.

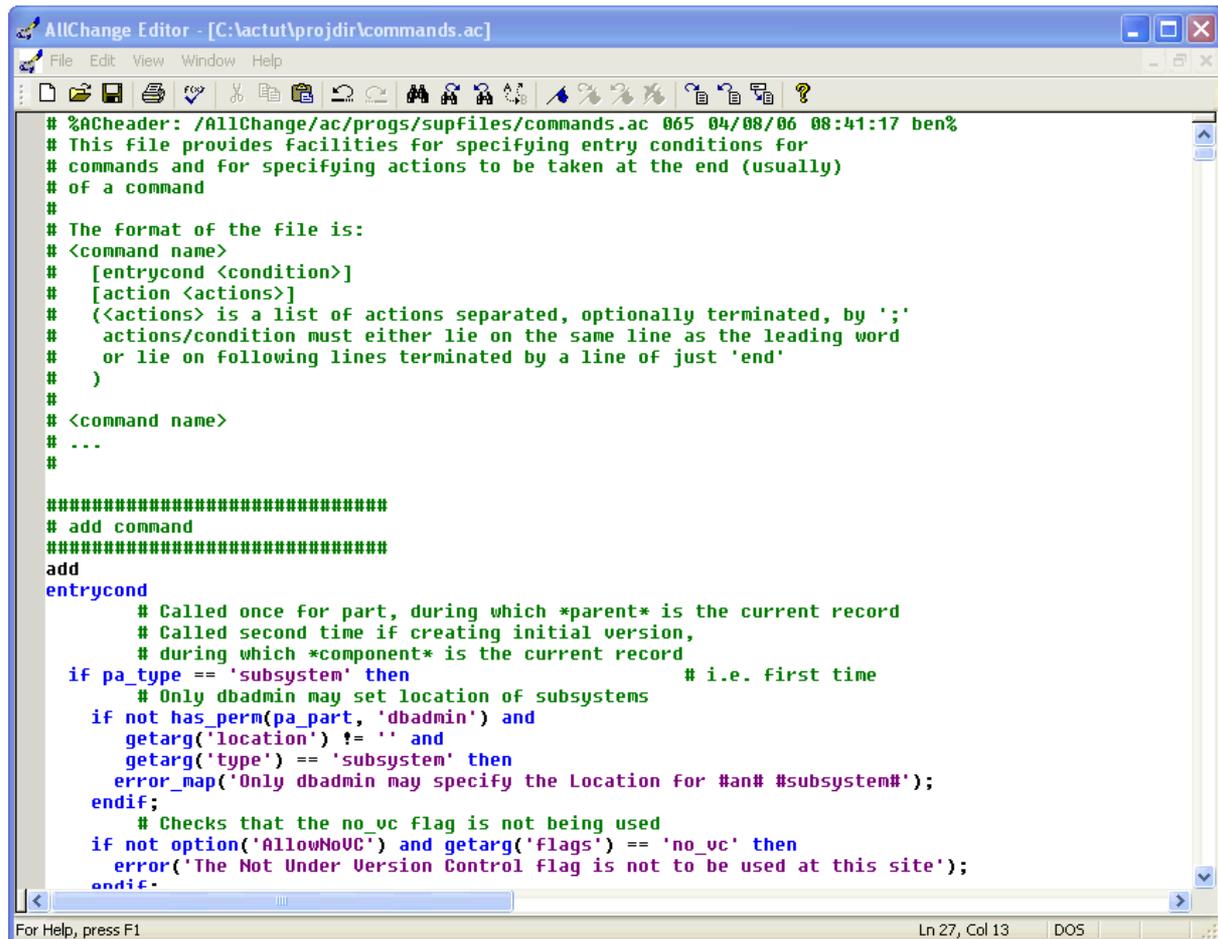
For full details of each configuration facility see [The AllChange Configuration Facilities](#).

### The AllChange Text Editor

#### The AllChange Text Editor - ACEdit

The **AllChange** Editor is called **ACEdit** and provides general text editor facilities together with syntax highlighting and checking for ACCEL.

It is used by ACCONFIG for editing some of the configuration files which contain large portions of ACCEL code and this is the most appropriate editor, e.g. Command definitions, Life-cycle conditions and actions and ACCEL function files.



**ACEdit** allows the current document to be edited providing that the file from which it was read is not marked as **Read Only**. If it is read-only then the status bar will show **READ** and no editing may take place.

**ACEdit** also supports syntax highlighting and syntax checking for known languages/formats. The built in ACCEL parser supports both syntax checking and syntax highlighting. Syntax checking may be invoked from the toolbar, the **File** menu or pressing F7. The syntax is also checked on saving the file. If the check

fails, the user is asked whether they wish to continue saving the file. This automatic check may be turned off in the **Options** dialog.

If a syntax check fails, then a message is displayed to the user describing the error, and a marker is drawn in the selection margin of the error line. Hovering the mouse over the marker displays the error description. The error marker may be cleared at any time by right-clicking on the error line and selecting **Clear Syntax Error**. Note that a maximum of one error is reported at any one time. Each error must be corrected before the remainder of the file can be checked.

Some **AllChange** configuration files do not use the ACCEL syntax but have their own format. The syntax checker may report errors for some of these files. To avoid this, files may be excluded from the checking by adding them to a list in the parser's options. The lists default to those files shipped with AllChange which cannot be parsed as ACCEL. To modify the list, open the View -> Options dialog, and click on the File Types tab. Choose the type of the file in the first drop-down list. The parser for the file type, if any, will be selected in the parser list. If the parser has options, then the Options button will be enabled. For the ACCEL Parser and Report Parser, this option dialog allows the list of files to ignore to be modified. (Note that the ACCEL Parser is separate from the Report Parser, and each has its own exclusion list.)

The status bar will also show the current line number and column position of the cursor.

The menu bar provides access to the facilities available in **ACEdit** with the most common options also being available from the toolbar.

**File Menu:** file and printing options

**Edit Menu:** facilities for searching and editing the current document

**View Menu:** user configurable display settings

## File Menu

The **File** menu provides access to file and print handling options.

### New

Creates a new *untitled* document which may later be saved to a file

### Open

Allows a file to be opened for viewing/ editing purposes. A standard **File Open** dialog will be presented to allow you to select the file that you want to open.

### Close

Closes the current document.

### Save

Saves the current file. If the current file has the read-only attribute then the save will be treated as a **Save As**.

### Save As

Saves the current document into the file name specified. If an existing file is specified you will be prompted as to whether you wish to overwrite the file. If a file is selected which has the read-only attribute set, then it will be made writable prior to saving the file, after confirmation to overwrite.

### Check Syntax

Performs a syntax check on the currently open file.

### Print

Prints the current document

### Print Preview

Shows a preview of what a print of the current document will look like

### Page Setup

Allows the page layout to be defined for printing

### **Print Setup**

Allows printing defaults to be specified

### **Exit**

Exits the editor

The file menu will also show the last few recent files. The number of files saved may be specified in **View | Options**.

### **Edit Menu**

The **Edit** menu provides facilities for searching and modifying the current document.

### **Undo**

Allows the last editing operation performed to be *undone*

### **Redo**

Allows the last *undone* operation to be re-applied as if it had never been undone.

### **Cut**

Copies the currently highlighted text to the clip board and then deletes it.

### **Copy**

Copies the currently highlighted text to the clip board.

### **Paste**

Inserts any text which has been copied to the clipboard into the current document at the current cursor position

### **Delete**

Deletes the currently selected text

### **Select All**

Selects the entire current document

### **Find**

Allows a string in the current document to be searched for. This will search for the specified text in the direction specified and will allow the search to continue when it reaches the beginning/ end of the document. Options supported include:

- **Match whole word only** ensures that only a complete word is matched for the text specified. e.g. a search for `fred` without whole word match selected will also match `freddy`
- **Match case** if selected will take into account the case of the search string
- **Regular expression** allows the search string to be specified as a regular expression.

Use **Find Next** to start the search for the next occurrence of the search string starting the search from the current cursor position.

### **Find Next**

Searches for the next occurrence of the previously specified search string.

### **Find Previous**

Searches for the previous occurrence of the last search string. Previous denotes that the search will take place in the opposite direction (up or down) from that originally specified.

### **Replace**

Similar to search but allows occurrences of the string found to be replaced with another string. The search/ replacement may be limited to the **Selection** or permitted to cover the **Whole file**.

Actions permitted include:

**Find Next** to locate the next occurrence of the search string

**Replace** to replace the currently highlighted result of the last find with the specified replace-

**Replace All** to replace all occurrences of the search text with the replacement text

### Advanced

The advanced menu provides access to facilities to:

**Make Selection Uppercase:** this will convert the selected text to all uppercase characters

**Make Selection Lowercase:** this will convert the selected text to all lowercase characters

**Sort Selection Ascending:** this will sort the selected text in ascending alphabetical order

**Sort Selection Descending:** this will sort the selected text in descending alphabetical order

**Transpose Characters:** this will *swap* the characters to the immediate left and right of the current cursor position

**Transpose Words:** this will *swap* the *cursor word* — this is the word within which the cursor lies (or the word to the right of the cursor if it lies on a *white space*) — and the word to the left of the *cursor word*.

**Cut Current Line(s):** this will remove the selected lines, or the line containing the cursor, and add the lines to the clipboard.

### Read Only

If selected makes the current document read-only so that no modifications may be made. This will be automatically selected if a file is opened which has its read-only attribute set. This setting may be changed to allow changes to a read-only file, or prevent changes to a writable file.

### Bookmarks

Allows book marks to be placed within the document and moved between. A bookmark will be shown in the selection margin if this is displayed. Functions include:

**Toggle Bookmark:** places/removes a bookmark at the current cursor line

**Next/Previous Bookmark:** moves to the next or previous bookmark

**Clear All Bookmarks:** removes all bookmarks

### Goto

The Goto menu provides access to:

**Goto Line:** allows a specific line number to be moved to

**Goto Matching Brace:** this menu causes the cursor to be moved to the brace paired with the brace at the current cursor position. This facility is available for round braces e.g. ( ), curly braces e.g. {} or square brackets e.g. [ ].

**Tag:** this menu offers the following facilities:

**Goto Tag:** if a tags file is available, jumps to the location defined for the current selection or the word at the cursor position. The current location is added to the tag stack so that it may be returned to by using **Pop Tag**. If more than tag entry is found for a word, then the user is presented with a dialog from where the desired definition may be selected.

**Prompt Tag:** as above, but a prompt dialog is always shown. This may be used if no word is selected, allowing the user to enter the word to tag.

**Pop Tag:** returns to the previous location tagged from, if any, and removes the location from the tag stack.

### View Menu

The **View** menu allows various options to be specified for the display of files in the editor and the configuration of the editor.

Options include:

### Toolbar

If selected the toolbar is displayed

### Status Bar

If selected the status bar is displayed showing various status information about the current document, including the current cursor location, and whether the file is currently read-only.

### Set Font

Allows the font used to be specified. The font selected is used only for display and printing, and is used by all files opened in the editor.

### Selection Margin

If selected the selection margin is shown. Bookmarks are displayed in the selection margin

### Show Whitespace

If selected then white space characters (e.g. tabs and spaces) will be identified with a visible character.

### Options

Allows various editor options to be selected concerning:

- the **Editor** for options which affect the editing facilities for a document
- the **Format** for the colours used for syntax highlighting.
- **File Types** for definitions of known file types and what syntax highlighting should be used. Also allows file types to be *registered* to use **ACEdit** to **open** files of that type (e.g. from Explorer)
- **Printing** allows header and footers to be specified for printing purposes

### Syntax Colouring

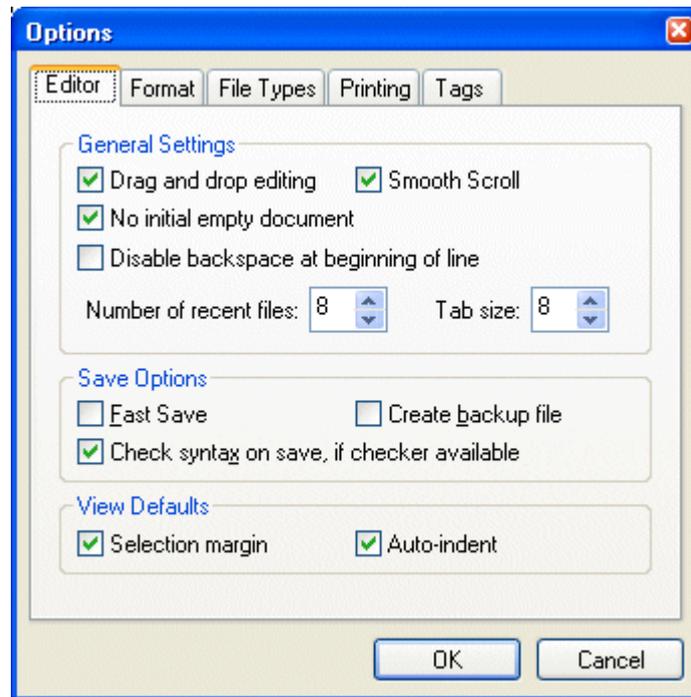
If selected and the file type for the current document is one known to **ACEdit** and which has a known syntax, then syntax highlighting will take place. If this is not selected then no highlighting will take place.

### Editor Options

The Options dialog allows various options to be changed, will control the way in which the editor is used. The options are divided between several tabs.

### Editor tab

The editor tab holds options which relate to the behaviour of the main editing window.

**Drag and drop editing:**

this option allows text to be selected and dragged to a new position.

**Smooth Scroll:**

this options provides a smoother scroll effect when scrolling large portions of the current file

**No initial empty document:**

the editor will open a new blank document on opening unless this option is enabled

**Disable backspace at beginning of line:**

this option prevents the backspace key from deleting text if the cursor is at the beginning of a line

**Number of recent files:**

this controls the number of recent files added to the bottom of the **File** menu.

**Tab size:**

this governs the number of spaces spanned by a tab character in a file

**Fast Save:**

This option causes file contents to be saved directly into the file. Without this option, files are saved to a temporary file, and, if the save is successful, the temporary file is copied to the final required destination. If permissions are set explicitly on the file being edited, rather than being inherited from its folder, then leaving this option turned off will result in the file's permission settings being lost. If the permissions need to be kept, then Fast Save should be used. If this is enabled, then the Create backup file option is not available.

**Create backup file:**

With this option enabled, saving a file creates a back-up file in the same directory as the saved file. The file is named the same as the saved file, but with an additional extension of '.bak' appended to the name. If this option is turned on, the Fast Save option is not available.

**Check syntax on save, if checker available:**

If the parser for the current file contains a syntax checker, this option will check the syntax on saving the file.

**Selection margin:**

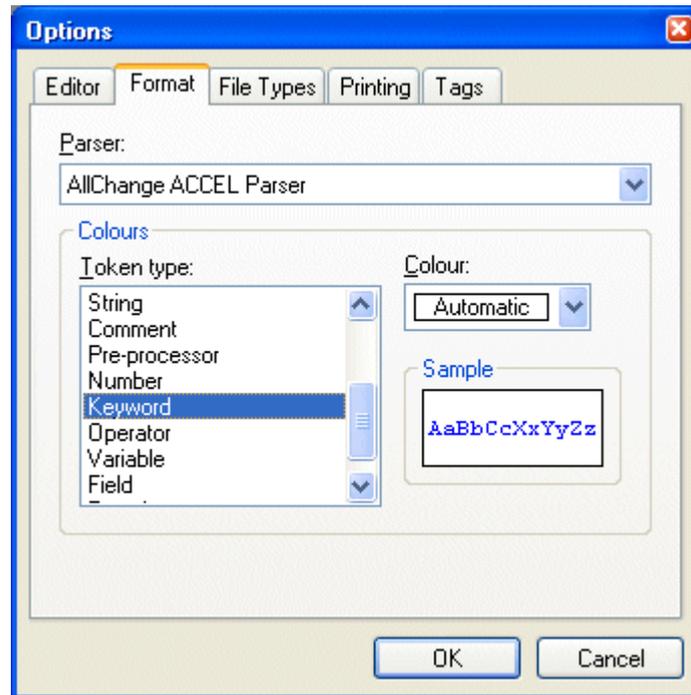
shows or hides the selection margin to the left of the editor window

**Auto-indent:**

with this option enabled, any whitespace characters (e.g. tabs or spaces) used to indent a line are copied to the new line when pressing the return key.

**Format Tab**

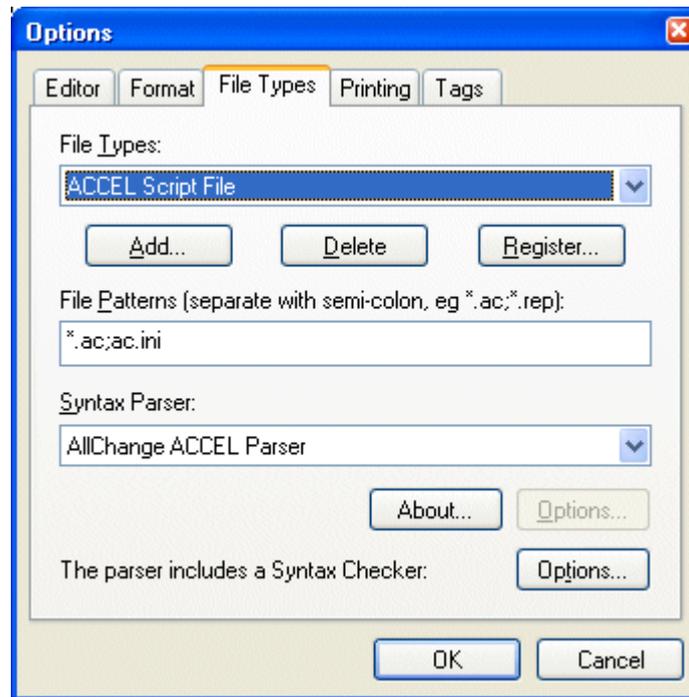
Syntax colouring options may be set on the Format tab for each available parser.



Selecting the Parser in the top list allows the colours to be set for each token type in a file using the parser. Setting the colour to Automatic will cause it to use the default colour for the token.

**File Types tab**

The file types tab provides options to link specified file types with the appropriate parsers, and allow parser options to be set.



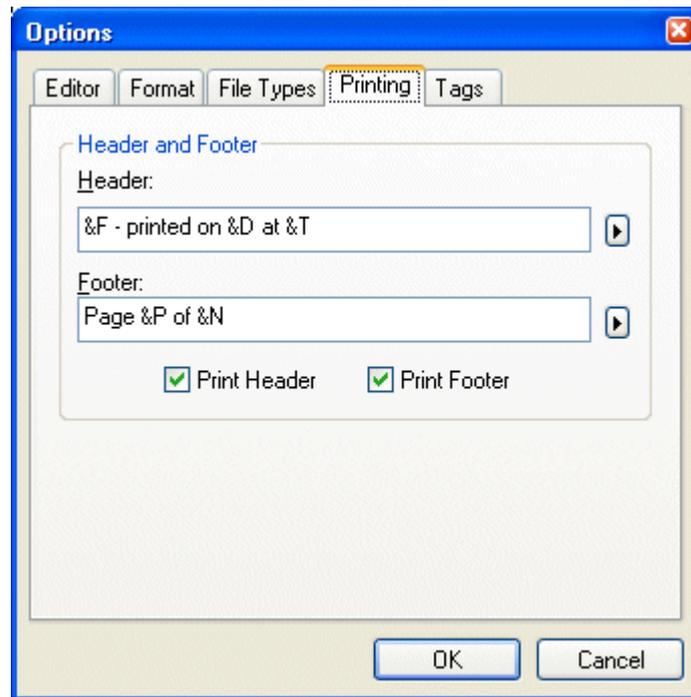
The file type may be selected from the top drop-list, or added by clicking "Add...", and the file's suffixes entered in the File Patterns control. The parser for that file type may then be selected in the Syntax Parser list. In the example above, files with a suffix of .ac, or called ac.ini, have been called "ACCEL Script File" and will be parsed using the ACCEL parser.

If you wish to register the file type with Explorer, so that opening a file within Explorer opens the file with the AllChange editor, then clicking on the "Register..." button allows the file type to be registered. The file type name to be shown in Explorer may be set here.

If the parser includes a Syntax Parser, then this is noted at the bottom of the tab. The "Options..." button leads to a dialog where options for the checker may be set. This allows a list of file patterns to be specified which are excluded from ACCEL checking. So, although a file such as roles.ac is registered here to use the ACCEL parser for syntax colouring, it does not have normal ACCEL syntax, and so may be excluded in the checker options.

### Printing tab

The printing tab allows various options to be set which affect how files are printed from the editor.



**Header:**

text can be specified here to appear at the head of each printed page. The text may include any text, along with replaceable codes to include information about the current document. The button to the right of the edit control allows these codes to be inserted. The choice of codes is as follows:

**&F - filename:** the full path of the current file

**&P - page number:** the number of the current page in the printed output

**&N - number of pages:** the total number of printed pages in the file or selection

**&T - current time:** the time at which the print operation is started

**&D - current date:** the date on which the print operation is started

**Footer:**

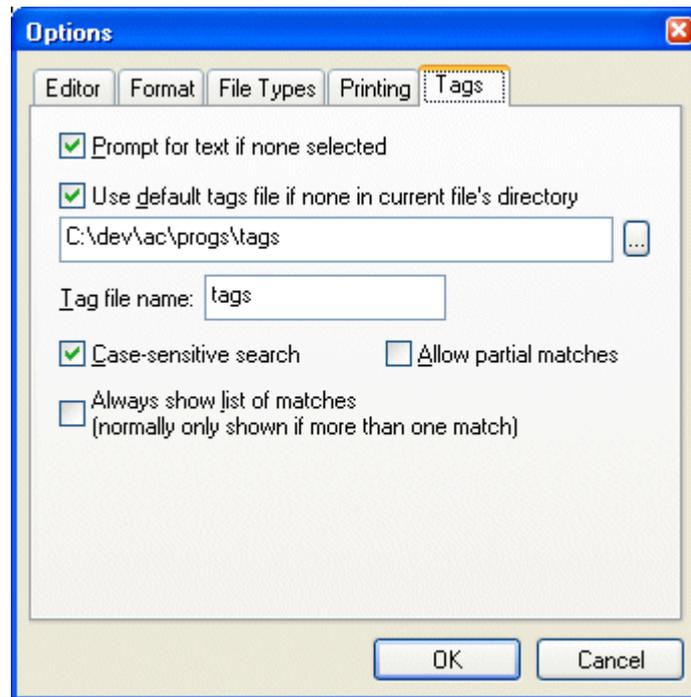
text for the footer for each page can be specified here. The text may include codes as described above.

**Print Header / Print Footer:**

specifies whether the header or footer should be printed. This allows the header or footer to be removed and re-instated at any time without having to edit the text.

**Tags tab**

The behaviour of the tags support can be modified from the Tags tab.

**Prompt for text if none specified:**

If no text is selected, then, with this option switch on, the user will be prompted for a token to search for in the tags file

**Use default tags file...:**

Normally the tag support will use a file named 'tags' in the same directory as the current file. To avoid needing to keep tags files in each file's directory, a composite tags file may be used, and its path entered here. If a tags file is found in the current file's directory, then that will be used in preference to the file specified here

**Tag file name:**

The default name for a tags file is 'tags' but an alternative name may be specified here

**Case-sensitive search:**

This option specifies whether the tag search should be case-sensitive. With this option enabled, "Text" will not be the same as "text".

**Allow partial matches:**

This option allows partial matches to be offered to the user in the list of matches. So, a tag of "FunctionName" will be offered when searching for "Function".

**Always show list of matches:**

With this option, the user is always presented with a list of matches for a tag. If not using this option, a list is only displayed if there is more than match.

## AllChange Projects

Before you can use **AllChange** you must create a project . To create your first project use ACCONFIG. When no projects are defined it will open the Projects window to allow you to create your project - see [Creating and Updating Projects](#). When you save your project definition it will automatically open the project.

You now have a basic project that you can then set up for initial use and later configure to your site's requirements.

You can now proceed and register your users, allocate roles to them and set up workspaces.

## Licensing Users

**AllChange** supports 2 different user licensing models:

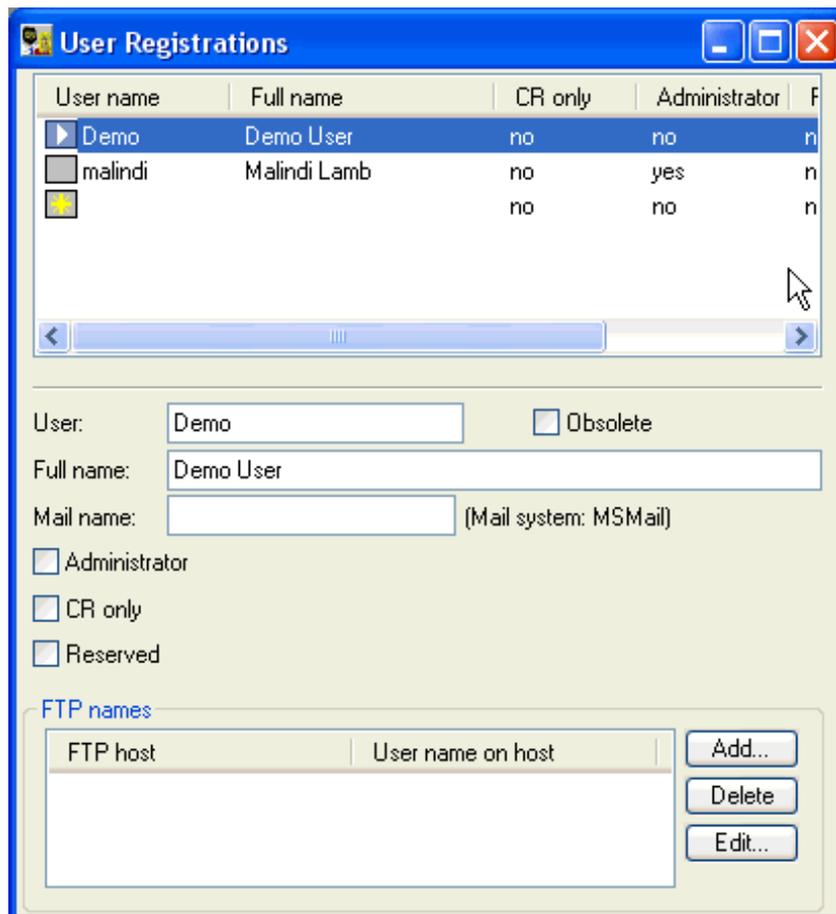
1. **Named user licensing.**  
An  $n$  user licence entitles up to  $n$  users to use the system.
2. **Concurrent user licensing.**  
An  $n$  user license entitles up to  $n$  instances of the system to be run at the same time. (Specified instances may be reserved for admin use)

For both these models normally a user's name *must* be known to the system as a registered user before the system will allow that user access to any **AllChange** program, however there is a facility to allow unlicensed users read only access to ACE. If unlicensed users are not permitted access (the default case) then this scheme also ensures that only authorised users can gain access to any part of the system, in addition to any further authorisation checks performed by individual commands, see [Licensing](#) for full details.

Before *anybody* can run any **AllChange** program (except ACCONFIG) they must be registered as an **All-Change** user.

The initial full installation gave you the opportunity to register yourself as an **AllChange** user.

Modification of the registered users is available from **Access | User Registration**



Initially, if no users are licensed, you do not need to be a licensed user to use ACCONFIG. As soon as one or more users are licensed, then ACCONFIG requires you to be a licensed user.

To modify details of an existing user registration select the user in the list and then make the required modifications to the details shown in the bottom half of the window.

To add a new user select **Edit | New Item**, or select the blank item at the end of the list, then complete the details in the bottom half of the window.

For each user you *must* give the user name as **User**, this should be the users' network logon ID, further details are given in [User Registration](#).

If a user who is not registered attempts to access **AllChange** an error will be issued (unless unlicensed access has been permitted).

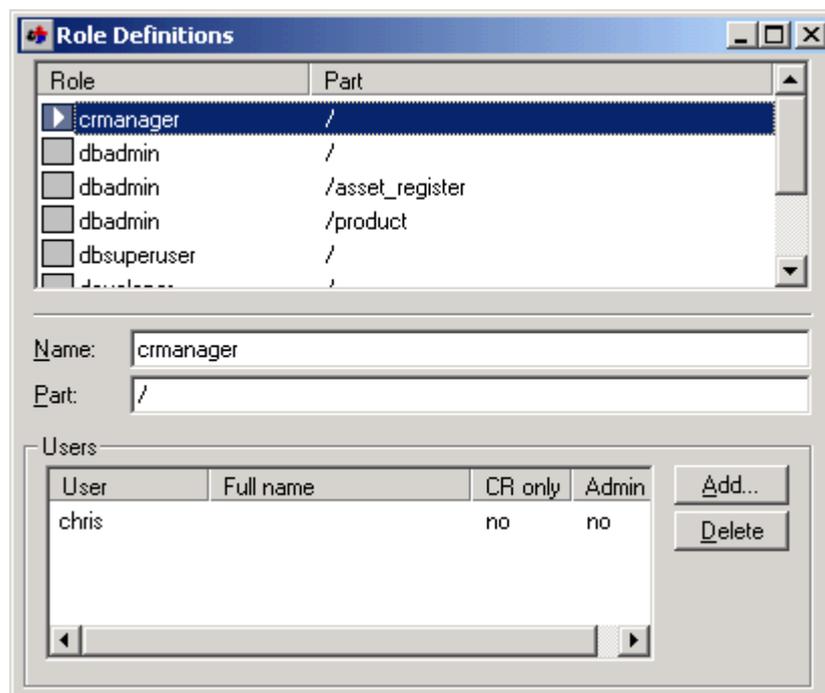
If using the named user licensing scheme then when the maximum number of licensed users is reached, ACCONFIG will prevent you from registering any further users. For concurrent user licensing there is no limit to the number of users that may be registered, only in the number that may invoke the application at one time.

To ensure that an Admin user does not find him/herself unable to log onto AllChange because all the concurrent licenses are in use by non-Admin users, **AllChange** provides a means of "reserving" licences for certain user(s). Select the **Reserved** option available in [User Registration](#) to specify a concurrent user license as reserved.

## Allocating Users to Roles

Roles are used to implement access restrictions within **AllChange**. Roles are defined to **AllChange** together with users permitted to have that role and a part associated with the assignment of that role. Roles effectively define groups of users with permission to perform various actions on certain areas of the parts database.

New roles may be defined, users may be assigned to roles and existing roles may be modified from **Access | Role Definitions** in ACCONFIG.



**AllChange** is supplied with five roles defined which are used in defining the *out-of-the-box* access controls. These are:

**crmanager:** users with this role may perform operations on CRs to which they are not necessarily assigned

**developer** users with this role may perform operations on parts (e.g. check parts in and out)

**release\_manager** users with this role may perform operations on baselines and release operations

**dbadmin:** users with this role may perform database administration tasks

**dbsuperuser:** users with this role may perform *any* task

**dbsuperuser** and **dbadmin** roles have a special significance in that a user is not granted these roles *unless* the role is specifically selected in ACE.

Each of the above roles is associated with *all* the configuration items in the parts database (i.e. has a **Part** of /), this means that the role allocations do not vary according to the parts being accessed.

In order to get started using **AllChange** you need to (at a minimum) allocate yourself (and anyone else as required) to these roles as appropriate. Use the **Add** button to add new users from the list of registered users. *Note that a user may have more than one role.* See **AllChange** User Manual for details on how roles are used in **AllChange**.

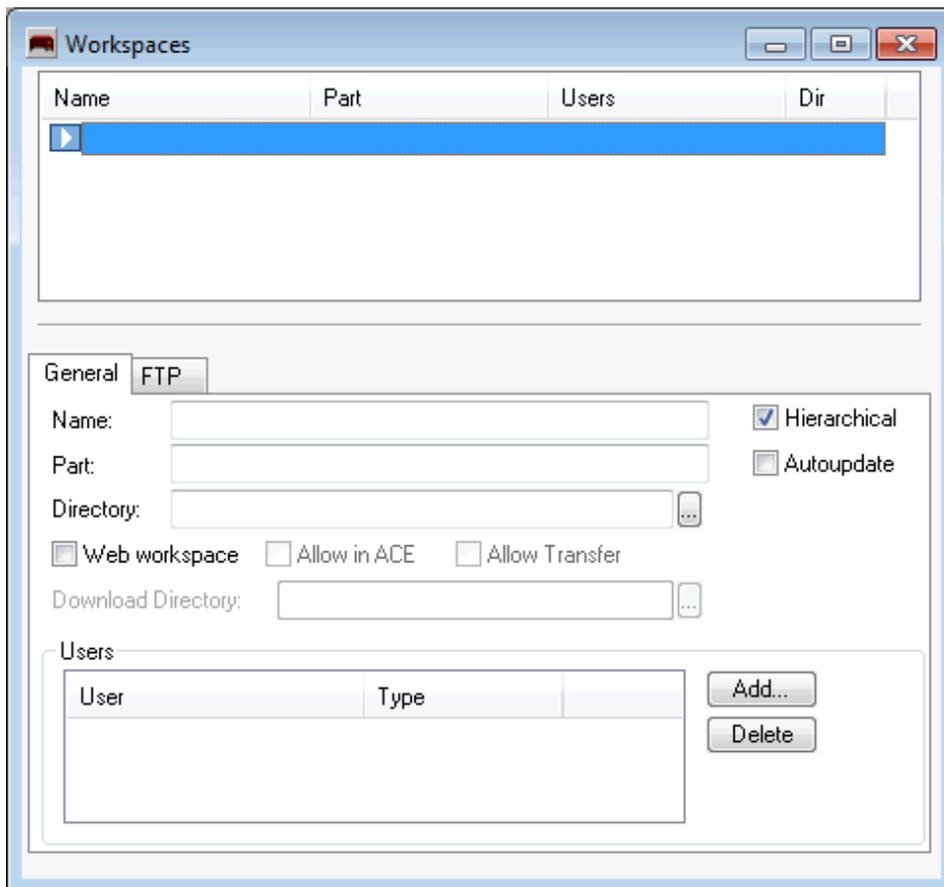
The roles as supplied are simply examples, you may wish to create new ones in order to implement the access controls required by your organisation — see [Defining Access Control](#) for further details.

## Defining Workspaces

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.

Before you will be able to import any existing files, or check any files in and out of **AllChange** you will need to define workspaces for the directories in which these files will reside. See the **AllChange** User Manual for details on how workspaces are used.

Workspaces may be defined and modified from **Workspace | Workspaces** in ACCONFIG.



Each workspace *must* have the following information:

### Name

An arbitrary name for the workspace. This 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. Initially you could define a workspace for the entire parts tree by specifying /, although ultimately you will want to define different workspaces for different parts.

**Directory**

The name of the physical directory where you wish the files to be placed (or the directory where the files currently exist). If the directory does not exist then you will be asked if you wish it to be created.

**Users**

The registered users who may use this workspace.

Workspaces may also have various other information associated with them details of which may be found in [Workspaces](#).

# Setting up your Environment

## About Setting up your Environment

There are a number of different factors which will affect how **AllChange** operates, this is referred to as the *environment* under which **AllChange** runs.

This chapter describes the different ways in which you can affect the levels of security, the performance and the facilities available to users by changing the environment.

The only environment information which **AllChange** requires under Windows before you can use it is the **AllChange** system directory . This is the directory into which you installed the **AllChange** software during a **Full Install** and contains various essential system files. This also includes a *template directory* where appropriate which is a subdirectory of the system directory containing configuration information for a particular project template

Your registry will be updated to provide this information on performing a workstation installation, see [Windows Registry Settings](#) for details of registry settings used by **AllChange**.

Other environment configurations are optional and you can implement as required.

Note that in addition to the **AllChange** system directory an **AllChange** project must have been created before ACE may be used, see [AllChange Projects](#). The definition of the project will, in turn, then define other information needed by **AllChange** such as the **AllChange** project directory (the directory containing the project configuration files and **AllChange** database).

## Defining the AllChange Environment

There are two basic methods for defining environment information to the **AllChange** tools: entries in Windows registry and operating system environment variables.

The **AllChange** install (workstation or full) will create a default set of registry entries for you based on the information given at install time.

The only environment variable/ registry entry which *must* be defined in order to supply the minimum required environment information is `ACDIR`. This must specify the **AllChange** system directory — the **AllChange** install program will specify this for you in the registry.

Environment variables/ registry entries have a *name* and a *value*: for example, the environment variable named `ACPROJECT` has as its value the name of the **AllChange** project which is to be used.

There are several possible sources of the various environment values that **AllChange** requires to operate. In decreasing precedence, these are:

1. Environment variable set via `-Evar=value` argument on the invoking command line
2. Environment variables inherited from the operating system
3. Registry
4. If not found in any of the above, a default value will be used where possible

When an **AllChange** tool needs the value for a particular variable it first looks to see if the value has been specified on the command line.

If no value has been specified on the command line then it looks for an appropriately named operating system environment variable — see Operating System Specifics for information on how these are set under individual operating systems. If such an environment variable exists **AllChange** uses its value. Environment variables are widely used under many operating systems for passing such information around, and are convenient to alter when required.

If the operating system environment variable is not found, **AllChange** tools then look in the Windows registry. If the appropriate subkey in the registry is found it is searched for an entry with the same name as the required environment variable.

If no suitable entry can be found **AllChange** will use a default (if appropriate). Note that defaults are overridden by registry entries, which in turn are overridden by environment variables.

Use of the command line is particularly useful under Windows so that various icons can be set up to call ACE with different settings without having to change the environment or registry. For example, the command line for an icon to invoke ACE client/server to a remote host could be:

```
\ACSYS\WIN32\ace.EXE -EACSERVER=remote
```

By use of the `-EACPROJECT=` command line argument various icons can be set up to run **AllChange** for different projects e.g.:

```
\ACSYS\WIN32\ace.EXE -EACPROJECT=intatest
```

The registry entries may be created by the installation program, by ACE or ACCONFIGOR by the supplied registry editor. In general you should not need to modify the registry yourself.

Further details of the registry entries used are in [Windows Registry Settings](#).

## SQL Server

**AllChange** uses Microsoft SQL Server as its database. This topic describes requirements, set up and functioning of SQL Server. The sub-topics are:

[Microsoft SQL Server 2008](#)

[Microsoft SQL Server 2008 Express Edition \(SQL Express\)](#)

[Installation](#)

[Creating an \*\*AllChange\*\* Project Database](#)

[Accessing an \*\*AllChange\*\* Project Database](#)

[SQL Server Permissions](#)

[Accessing the Database Outside of \*\*AllChange\*\*](#)

[Relationship Between SQL Server and \*\*AllChange\*\* Server](#)

### Microsoft SQL Server 2008

**AllChange** works with Microsoft SQL Server 2008 R2 onward (note that this is the "R2" Release), any edition, 32- or 64-bit.

SQL Server Books Online ("BOL") provides all documentation related to Microsoft SQL Server. **AllChange** documentation does not attempt to cover all the ground for SQL Server, only what is relevant to the application. See BOL for anything further.

The system requirements for SQL Server 2008 vary according to edition and operating system. See Microsoft documentation for the latest requirements.

SQL Server must be installed on a machine which is to act as a server for the **AllChange** database. Typically, this is a *different* machine from the one(s) where **AllChange** applications are run from (i.e. end user workstations). SQL Server does *not* need to be installed on client machines, only on the server.

If administering SQL Server directly, **SQL Server Management Studio** ("SSMS") offers functionality to administer databases.

### Microsoft SQL Server 2008 Express Edition (SQL Express)

Microsoft SQL Server 2008 (onward) is required as the database engine for IntaChange. Those who wish to run a cut-down version of SQL Server with **AllChange** may use Microsoft SQL Server 2008 Express (supplied with **AllChange**) instead of using the full SQL Server. Microsoft provides SQL Express as a freely redistributable, cut-down of the SQL Server 2008 data engine.

If you already have (any edition of) SQL Server 2008 (onward) available for use from workstations where **AllChange** has been installed, you will (presumably) wish to use this. If so you may need to contact your SQL Server administrator to determine the name of the SQL server and the username/password required to connect to it.

If you do not have a suitable edition of SQL Server, **AllChange** is supplied with Microsoft's SQL Server 2008 Express (32-bit, works on both 32- & 64-bit machines). This is a freely redistributable version of the SQL Server 2008 data engine, supporting most of the functionality of the other editions of SQL Server 2008, but optimised for use on smaller computer systems. It is possible to upsize seamlessly from SQL Express to full SQL Server if your requirements grow; SQL Express may also provide a simple evaluation resource for **AllChange** without needing to involve a site's full SQL Server implementation. When creating a new project there is a [Use settings for AllChange-installed SQL Express](#) checkbox to use it as installed.

A "vanilla" version of SQL Express is supplied with **AllChange**. There are also various SQL Express *Advanced Services* Editions, which are still free to use and downloadable from Microsoft web site (e.g. <http://www.microsoft.com/downloads>). We would recommend any site which intends using SQL Express seriously to install that edition, as it contains additional features (in particular **SQL Server Management Studio**).

At the time of writing, the requirements for this supplied SQL Express 2008 R2 are:

- Processor: P3 1.0 Ghz (2.0 Ghz or faster recommended).
- Operating System: Windows XP SP3 onward (including Server 2003 SP2, Vista SP2, Server 2008 SP2, Windows 7, Server 2008 R2).
- RAM: 256 Mb minimum (1 Gb recommended).
- .NET 3.5

## Installation

There are two aspects to SQL Server installation:

- SQL Server itself must be installed on a machine, to provide database services.
- Any other machines which require access to SQL Server (i.e. end user workstations) need client-side installation of components.

### *Server Installation*

SQL Server must be installed on whatever machine is to act as the database server. That machine does not even require **AllChange** to be installed, just SQL Server. However, it is usual (but not required) for that machine to be the one where ACSEVER is installed.

There are two typical scenarios:

- An existing edition of full SQL Server is to be used. This has already been installed on a server machine, by SQL/Network Administrators. In this case, you will need the Administrators to supply information/access when you come to create **AllChange** projects (not at installation time).
- The supplied SQL Express is to be used. (This is often the case in "evaluation" scenarios.) This will be installed while **AllChange** is installed.

When the **AllChange** Setup program performs a *server* installation phase, it offers a choice between installing the supplied SQL Express or electing to use an existing SQL Server of your own (in which case no SQL Server installation is performed).

If you have more than one **AllChange** project, it is possible to use different SQL Servers/instances for each project's database, if desired.

## SQL Server Instances

SQL Server allows the installation of individual *instances*. These act effectively as separate installations. The advantage is that they are isolated from one another, allowing separate databases, permissions, uninstallation etc. The disadvantage is that the SQL Server installer must be run separately to install/patch each one.

If you elect to install the supplied SQL Express it will create an individual instance (named **AllChange**) for use by **AllChange** alone. It will not interfere with any other SQL Servers or SQL Expresses which are or might be installed on the same server.

If you wish to use your own SQL Server for **AllChange**, consider creating a dedicated **AllChange** instance for it. Although not compulsory (you may use a default or any other instance), it may be advisable. Discuss with your SQL Administrator.

## Client Installation

Any client machine (end user workstation) running **AllChange** will need to access the SQL Server. **AllChange** uses ADO/OLEDB routines, with *Microsoft SQL Server Native Client* as the provider.

When the **AllChange** Setup program performs a *workstation* installation phase, it detects whether a suitable SQL Server Native Client is installed and offers to install the necessary components if it thinks they are not present.

Any client machine for **AllChange** administrators will also require the **SQL Command Line Utilities Component** (SQLCMD & BCP are required by ACCONFIG to administer databases). If the client machine has SQL Server (including Intasoft supplied SQL Express) installed on it, these utilities will be there. However, if the SQL server is on another machine, these utilities need to be installed on the administrator workstation(s).

When the **AllChange** Setup program performs an administrator workstation install (including during a Full install), it detects whether suitable tools can be found and if not offers to install them. See also [Supplied SQL Script Files](#).

## Creating an AllChange Project Database

Each **AllChange** project must have its own SQL database. ACCONFIG is used to create a new project and it will create a corresponding SQL database, see [Creating and Updating Projects](#).

## Supplied SQL Script Files

When ACCONFIG performs actions in SQL Server such as creating/renaming/deleting/backing up databases, it uses SQL scripts and command batch files to do this. The **.sql** and **.bat** files are supplied in a sub-directory of the **AllChange** installation area named **Database** (or in an **Upgrade\_8\_1** sub-directory of **Database** when upgrading from **AllChange** 8.x only). You might need to inspect these, and edit if there are any problems.

The SQL tools SQLCMD.EXE and BCP.EXE must be available for use by **AllChange**. **AllChange** will use the environment variable SQLTOOLSPATH if it exists to specify the path to the tools directory otherwise it will assume the tools are available on the current path.

If you have more than one version of SQL Server installed and you are using the path, then you should ensure that the tools directory for the correct version of SQL server (i.e. that used by AllChange) comes first on the path. If not you should either modify your path or set the SQLTOOLSPATH environment variable to the correct location.

## Accessing an AllChange Project Database

When an **AllChange** program (ACE, ACC, ACCONFIG) accesses a project's database it must establish a direct connection to SQL Server. It uses the information about SQL Server name/instance, authentication and database name saved by ACCONFIG in the **projects.ac** file. The application must be able to access the

SQL Server/instance, e.g. there must be no firewalls blocking it from the client machine. By default, SQL Server uses TCP/IP port 1433 for connections (but only to a default instance, not a named one, which will instead be assigned a dynamic one), or 443 if using SSL encryption.

To successfully connect, a client must be *authenticated* by SQL Server. SQL Server offers two different possibilities:

- [SQL Server Authentication](#).
- [Windows Authentication](#).

### SQL Server Authentication

When using SQL Server Authentication, the database is accessed via a username/password, which has been added to SQL Server. *The actual Windows identity of any user accessing this way is not known/irrelevant to SQL Server.* No action is necessary in SQL Server to allow any users access.

The **AllChange** Administrator, when using ACCONFIG to set up projects, needs to know the SQL username and password. If using a SQL server on the network, your SQL Administrators will need to set this up and inform the **AllChange** Administrator what to use. Having typed these credentials into ACCONFIG, when they are saved into the **projects.ac** file the password is stored *encrypted*. Hence a user who examines that file cannot determine what the actual password is, and consequently cannot access the database directly outside of **AllChange**. The **AllChange** Administrator should never let end users know what the password is.

The advantages of SQL Authentication are simplicity (no need to do anything no matter what users access **AllChange**) and security (users have no direct access to the database, they can only go through **AllChange**). The disadvantage is that the security relies on a single password for access (anyone discovering that password can access the database directly, with all the permissions of the SQL user).

If using the SQL Express supplied with **AllChange**, and checking the **Use settings for AllChange-installed SQL Express** checkbox, SQL Server Authentication will be used. If you use this in a production environment, you should change the password (and possibly the SQL user too) from that supplied, so that end users cannot know what it is. To do that you (or a SQL Administrator) will need to use SSMS. Having done so, any projects which are currently using **Use settings for AllChange-installed SQL Express** checkbox must be updated to no longer do so, and the new username/password typed in and saved.

### Windows Authentication

When using Windows Authentication, the database is accessed *under the current user's logged-in Windows credentials*. Action is necessary in SQL Server to allow users access in this way, either via their explicit individual username or (more commonly) via an Active Directory group to which they belong/have been added. You (or a SQL Administrator) will need to use SSMS to accomplish this.

Windows Authentication to SQL Server is also known as *Trusted Connection*.

The advantages of Windows Authentication are no single point of password access (rather, access is as secure as the Windows logon), plus the possibility of setting distinct SQL Server permissions on a per-user basis. The disadvantages are that for a user to access a database the Administrator must either add that user into SQL Server or ensure he belongs to an already-added group, and the user could use some other tool than **AllChange** (e.g. SSMS) to access the database directly with all the permissions he has in **AllChange** (e.g. a normal **AllChange** user obviously needs add/update/delete access, so then he can go making any changes to the data he wishes, quite outside of **AllChange's** sanity/permission checks).

If using the SQL Express supplied with **AllChange**, you can uncheck the **Use settings for AllChange-installed SQL Express** checkbox and select Windows Authentication if you wish, *provided you are logged in as the user who installed the SQL Express*. This is because the installation set the current user as a SQL Administrator. No other user would then be able to access the database until they had been added into SQL Server with appropriate permissions.

## Authenticating from the Web Client

When users connect to **AllChange** via the web client, the web server code executes an ACC on each user's behalf which services the requests. This ACC is invoked from the ACINETD Windows Service under the account that service is running as, *not* any Windows account which the user might or might not have. If SQL Server Authentication is being used the account is irrelevant, but if Windows Authentication is used that means it is the Windows Service account which must be granted access to the SQL database. If required, you may alter the account under which this executes, e.g. you may need to change it to some domain account.

## Troubleshooting SQL Server Connection Issues

SQL Server connection issues/errors may arise at any of the following points:

- Populating SQL Server or database names drop-downs.
- Testing the connection.
- Performing database actions within ACCONFIG.
- Connecting to the database from ACE/ACC(end-users).

Here are some the problematic behaviours/messages which might be encountered:

**SQL Server name** does not show some SQL Servers/instances

**Database Name** does not show some databases

Error info includes: *Could not open a connection to SQL Server.*

Error info includes: *SQL Server Network Interfaces: Error Locating Server/Instance Specified.*

Error info includes: *SQL Server Network Interfaces: Server doesn't support requested protocol.*

Error info includes: *TCP Provider: No connection could be made because the target machine actively refused it.*

Error info includes: *Cannot open database "... " requested by the login. The login failed.*

Error info includes: *Login failed for user '...'*

- Remote SQL Server machine(s) not running SQL Browser Windows Service.
- Remote SQL Server machine's firewall is blocking UDP Port 1434.
- Client is on slow connection (WAN/VPN).
- Selected SQL Server/instance does not exist.
- Selected SQL Server/instance does not have its Windows Service running.
- Selected SQL Server/instance does not have TCP/IP protocol enabled.
- Selected SQL Server machine's firewall is blocking TCP/IP access.
- Authentication credentials are incorrect for accessing SQL Server/instance.
- Selected SQL Server does not exist.
- Selected SQL Server/instance does not have its Windows Service running.
- Selected SQL Server does exist, but named instance does not.
- Selected SQL Server/instance does not have TCP/IP protocol enabled.
- Selected SQL Server/instance does not have its Windows Service running.
- Selected database does not exist in SQL Server.
- SQL Server Authentication: the SQL Server user name or password is incorrect.

Error info includes: *SSL Provider: ....*

- Windows Authentication: the current Windows user has not been added to the SQL Server's logins for Windows Authentication.
- There is a problem of some kind with encrypted connections between client and server.

### SQL Server Permissions

Whichever SQL Server Authentication method is used, the account connecting to SQL Server will require certain permissions. In the simplest case, many sites are happy to use the **sa** account which has full permissions. If using the SQL Express as supplied with **AllChange** this will be the case.

For those sites wishing to be more restrictive on the permissions required, the *SQL Server Roles* required by **AllChange** are as follows:

<b>public</b>	Required.
<b>dbcreator</b>	Required in order to create the database. Can be removed after project creation.
<b>sysadmin</b>	Required in order to backup the database.
<b>bulkadmin</b>	Required during database creation <i>if upgrading from AllChange 8.x</i> (only). Can be removed after project upgrade.

### Accessing the Database Outside of AllChange

Although it is possible to access the SQL databases used by AllChange outside of the AllChange applications (e.g. from ODBC, SSMS, or a SQL reporting tool), this is unsupported. Intasoft does not support any kind of direct access to the SQL database which does not come from the AllChange application itself, and strongly recommends that you do not attempt to do so.

If attempting to read data from the database, as is common with complex applications although the tables contain the raw data much of the rules/interpretations/interrelationships/structures is effectively contained in the **AllChange** programs' logic. That will be lacking in any direct access to the data, and consequently the data may not "make sense".

If attempting to add/delete/update any data in the database, it is *extremely* likely that without the **AllChange** program logic things will go wrong or become inconsistent. You are in grave danger of losing information, or even making the whole project inaccessible, if you do this.

Furthermore, be aware that any per-user/role permissions to write or read data (e.g. ACCONFIG Command Access on updating or reading commands) are quite unknown to SQL (they are far too complex). Using any external tool will totally by-pass any **AllChange** permissions.

The only exceptions are for administering the database as outlined (e.g. performing backups, or changing authentication permissions), or if directed to do so in conjunction with Intasoft Support.

*Please note that Intasoft Support for AllChange does not cover discussion of the SQL data structures, access from external tools or general SQL Server questions.*

### Relationship Between SQL Server and AllChange Server

There are two different "data servers" used by **AllChange** applications: SQL Server and [AllChange Server \(ACSERVER\)](#), each hosting data and providing services. Both act as "data servers", in the sense that applications do not access the data directly but rather by exchanging messages (via sockets/ports) with processes running on remote servers. An **AllChange** client (ACE, ACC) holds quite separate connections (sockets/ports) to each of these.

**AllChange** uses a SQL Server database to store much of its end-user accessible data, e.g. information about CRs, baselines, parts, issues etc. However, because of the scope of **AllChange** this is far from *all* the data it stores — that would be far too large or too unwieldy to store in SQL Server.

**AllChange** uses ACSERVER to store a range of other data, in external files — including the Version History files, attachments and configuration files. (ACSERVER also provides certain non-data *services* to clients, e.g. remote file manipulation, common server time.)

There is no relationship between the SQL Server and the **AllChange** Server (at least as far as they are concerned). They do not know about each other or access each other. **AllChange clients** direct requests to whichever one is appropriate, and it is those clients which ensure that data in each is kept consistent. For example, when a user checks in a new version of a workfile, calls are made to the SQL Server to query and update the Versions and Issues tables, while calls are made to the **AllChange** Server to store the workfile contents into the corresponding Version History file.

In the most straightforward — and recommended — scenario, one physical server machine supplies both the SQL Server and **AllChange** Server (together with their data files). This keeps the situation simple.

However, **AllChange** does not *require* that these two servers be on the same physical machine — it is permissible to use distinct servers. Some sites have requirements for this, e.g. they may wish to keep all SQL Servers, and nothing but SQL Servers, on a dedicated machine. For efficiency, the only important thing is that each of these servers be as close as possible to where they physically store their data on disk, i.e. it is usual for SQL Server to use a local drive for its databases and for **AllChange** Server to use a local drive for all the files it accesses.

If you do elect to use separate machines for SQL and **AllChange** Servers, bear in mind that it is very important to keep the correct entries for these in tandem in ACCONFIG (i.e. the server(s) used in each project's **Database** and **Server** tabs). (It is of course equally important that they refer to the *same* machine if you are using one to host both servers.)

Finally, regardless of server location(s), it is just as important that the SQL database (name) is the correct one to use with the **AllChange** Server's project directory, i.e. the right database is being used with the right project directory. When creating projects, we suggest you adopt a common name for the project, its SQL database *and* its ACSERVER project directory.

## Modes of Operation

### About Modes of Operation

There are four possible basic modes of operation that **AllChange** supports. These involve different required levels of access to the **AllChange** system/ project files and to the VC files. These are (in increasing order of complexity/ sophistication):

1. no client/server operation ("direct" access) — see [Direct](#)
2. client/server access to **AllChange** files (but not the VC files) ("client/server" access) — see Sections [client/server](#) and Client/ Server Operation, with Direct Access to VC Files.
3. client/server access to **AllChange** files, plus client/server access to the VC files ("VC client/server" access) — see Sections [client/server](#) and VC client/server

Briefly, here are the advantages/ disadvantages of each mode:

#### Direct

Simplest to set up; least efficient; least secure; no requirements for server.

#### client/server

Improved security for **AllChange** system and project files; no change to efficiency or security of VC file access..

#### VC client/server

Same security for **AllChange** project and system files as client/server mode; improved efficiency and security for VC file access; most complex to set up; most efficient and secure.

Note that client/server modes of operation affect how **AllChange** project/system files and (possibly) VC files are accessed. They do *not* affect how end-user workfiles are accessed: in all cases direct access to these from an **AllChange** client is required, regardless of whether workspaces are physically located locally on the client or in some (possibly shared) area on a server. If workfiles are located on a server, there

is no requirement that this be the same server as that used for **AllChange** or VC directories. See [Security and Access Issues](#) for a detailed discussion of the access requirements to various files/ directories in different modes of operation.

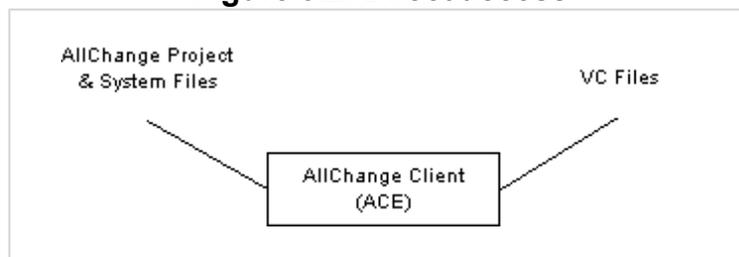
Note too that only end-user **AllChange** clients — ACE and ACC— can run in client/server mode. Other **AllChange** programs, such as ACCONFIG, *always* require direct access to the files they access and cannot be run client/server.

Each mode of operation is discussed in detail in subsequent sections.

## Direct

[Figure 3.2](#) illustrates the basic direct access, non-client/ server model.

**Figure 3.2: Direct access**



This is the most straightforward mode of operation. It is also used by administrator **AllChange** client programs, even if end-user **AllChange** client programs (ACE) do run client/server.

Direct access, via a network, is required to *all* files: **AllChange** database and support files in the project and system directories, and VC files. The end-user will normally need full permissions to most directories — see [Security and Access Issues](#).

The term "server" may still be used in this context: it refers to the machine which holds the **AllChange** files, which would usually be some kind of network server. (It is also perfectly possible for the client to be the "server" too, e.g. when using a "playground" area.) Since no **AllChange** programs run on the server there are no operating system or platform requirements, provided it supplies the client with native file access. **AllChange** has been used in this manner with a variety of operating systems and network software, including Microsoft, Novell, NFS, PC-NFS and Samba Networking.

Note that it *is* possible to set up and start using a system in direct mode initially for simplicity, and then move on to a client/server mode at a later date.

## Client/Server

### Setting up client/server — Summary

To install and setup to run client/server for Windows Server and Windows clients assuming a full install has already been performed:

1. Run the **AllChange** install program on the server (setup.exe in the setup subdirectory of the **AllChangesystem** directory ).

*You will need to be logged on as an administrator on the server itself.*

Select the **Server Only** installation option. This will ask you to provide the following information:

**Remove previous AllChange version's registry settings:** this will be disabled if the previous version did not make use of registry settings. If selected then the settings for the all projects for the previous version of **AllChange** will be removed.

You will then be asked if you wish to install the **ACInetd** now, or whether you will do it later manually. If you select to do it now then the following additional information should be specified:

**Service name:** this should be blank unless the default service name is not to be used. This need only be completed if it is required to run two different services for two different **AllChange** versions (e.g. when upgrading)

**Service display name:** this should be blank unless the default service display name is not to be used. This need only be completed if it is required to run two different services for two different **AllChange** versions (e.g. when upgrading)

At the end of the installation appropriate entries in the Windows registry will have been created containing the information required by the server.

2. If you did not install the **ACInetd** during the install in the previous step you should install it (run `acinetd -install` from the **AllChange** executable directory using a Command Prompt) — see [Configuring ACInetd for Windows NT](#).
3. Configure the **ACInetd** — it is recommended that it is configured to use a special **AllChange** account in which case you will need to create the account (e.g. **AllChange**) e.g. using the Windows Administrator tool **User Manager for Domains**.

You should then use the Control Panel **Services** to configure the **ACInetd** to start manually or automatically as required, and to log on as the **AllChange** user if appropriate. See [Configuring ACInetd for Windows NT](#) for details.

4. Set up the directory permissions. It is recommended that the following groups of users are created:
  - **AllChangeUsers:** all **AllChange** users should be members of this group (optional)
  - **AllChangeAdministrators:** all **AllChange** administrators should be members of this group

The minimum recommended permissions shown below may then be set up using the standard Windows file/directory permissions facilities:

- **AllChange**project directory, all files: **AllChangeUsers** should have at most read access. **AllChange** Administrators should have full access. **AllChange** user should have full access.
- **AllChange** system directory, all files: **AllChangeUsers** should have at most read access. **AllChange** Administrators should have full access. **AllChange** user should have read access.
- **AllChange** executable directory (WIN32 subdirectory of the system directory), all files: **AllChangeUsers** should have at most read and execute access. **AllChange** Administrators should have full access. **AllChange** user should have execute access.
- All subdirectories of the project directory and recursively down as files/directories are created: **AllChangeAdministrators** user should have full access.

If planning to run in VC client/server mode then **AllChangeUsers** require no access and the **AllChange** user requires full access.

If not planning to run in VC client/server mode then **AllChangeUsers** require full access and the **AllChange** user requires no access.

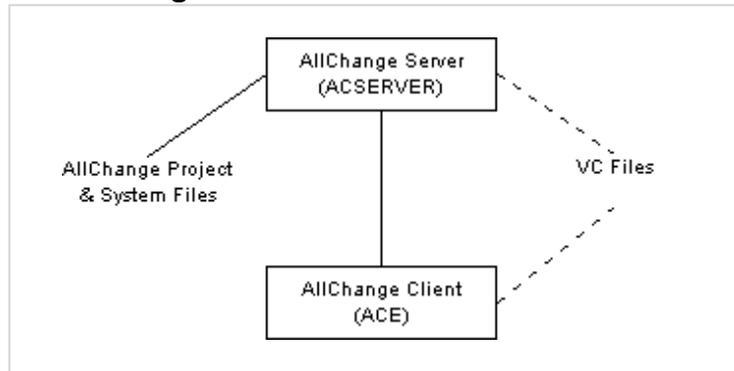
5. If you wish to run in VC client/server mode of operation (recommended), enable this using in the **Configuration Options** in `acconfig` (see [VC Client/ Server](#))
6. Use `acconfig` to set up the appropriate client/server information for your project(s).
7. Run the **AllChange** install program on each client (`setup.exe` in the setup subdirectory of the **AllChange** system directory). Select to perform either a **User** or **Administrator Workstation** install as appropriate.

### About Client/Server

AllChange may be configured to access project and system directory files with a client/server model. *Note that this has no effect on the database access which is provided by SQL Server.*

[Figure 3.3](#) illustrates the basic client/server access model. Note that the access requirements for VC files depend on which variant of the client/server model is adopted, as detailed in subsequent sections.

**Figure 3.3: client/server access**



There are different variants of client/server operation but all variants provide client/server access to the **AllChange** project and system files; they differ only in how access to VC files is gained. The information in this section is common to all variants.

The **AllChange** server program is called ACSERVER and is used in all variants of client/server operation.

There are two benefits to using client/server facilities:

- efficiency
- security

The precise efficiency improvement to be expected from client/server operation depends on many factors: the speed of the server, the underlying network, what operations are regularly performed in **AllChange**.

All client/server variants offer improved security for access to the project and system directory files, compared with non-client/server. Client/server operation with direct access to VC files offers no improvement in security or efficiency for accessing VC files; VC client/server offers improved efficiency and security for accessing VC files. Improved efficiency when accessing VC files will be noticeable when checking files in and out of **AllChange**.

The **AllChange** server is available for Windows 2003 Server SP2 or later.

The following **AllChange** clients are available:

- Windows 2003 SP2 or XP SP3 or later (Windows GUI and Command line)

The system may be configured to run one of the servers and any combination of the clients in client/server mode of operation, within the restrictions explained below in each variant.

A client/server architecture is supported for user-level **AllChange** programs (ACC, ACE) which provide the standard socket interface across TCP/IP (a PC TCP/IP must provide a Windows Sockets interface, e.g. Microsoft TCP/IP-32 which is supplied with Windows). Administrative **AllChange** programs (ACCONFIG) do not run client/server.

The project directory *must* reside on the same machine as the **AllChange** servers are to be run. One server will be invoked for each client **AllChange** session (this may be performed manually or automatically).

For each **AllChange** program running on a client one **AllChange** server process runs on the server machine to provide all services for the client program; messages are exchanged via the socket mechanism across TCP/IP. The servers must run on the same machine as the database is held. All database accesses are therefore local to the server with the overhead (compared to single machine access) lying in the request/reply message passing, which may prove superior to straight network access. Note that network access (e.g. NT, Novell, NFS, PC-NFS or other network product) to the project directory is not required but may be found useful. Network access to VC files may or may not be required, depending on which variant of client/server is in use. Network access to work files is *always* required.

In client/server operation, the actions log file `acacts.log` any system files which are sought in the project directory and most system files which are sought in the system directory (see Sections [Security and Access Issues](#) and [System Files](#)) are accessed client/server. The exception to this is the project definitions file, `projects.ac`, which is accessed directly. (Remember that only ACE runs client/server, not ACCONFIG which will still require direct access to these files.) A system file which is sought/ found elsewhere — e.g. in the current, home or Windows directory — is not accessed client/server; direct access to such files is required.

### Client/Server effects on reading of Time

When running client/server, **AllChange** reads the current time from the server instead of the client (where appropriate). This addresses problems where multiple clients have their clocks set inconsistently. For example, status logs store the current time and are sorted on this, so when using client times it is possible for status changes performed from different clients to appear in a different order from that in which they were actually executed if the clients' clocks were sufficiently inconsistent. Using the server time assures consistency.

Note that the client's clock is still used in certain situations. For example, if a workfile is saved on a local drive its modification time will be set from the client's clock. This time stamp will be preserved in the VC file when the file is checked in, and restored on check out. Local times are also used during the Build process. Consequently we still strongly recommend that clients have their clocks set accurately. If not running client/server local client time is always used.

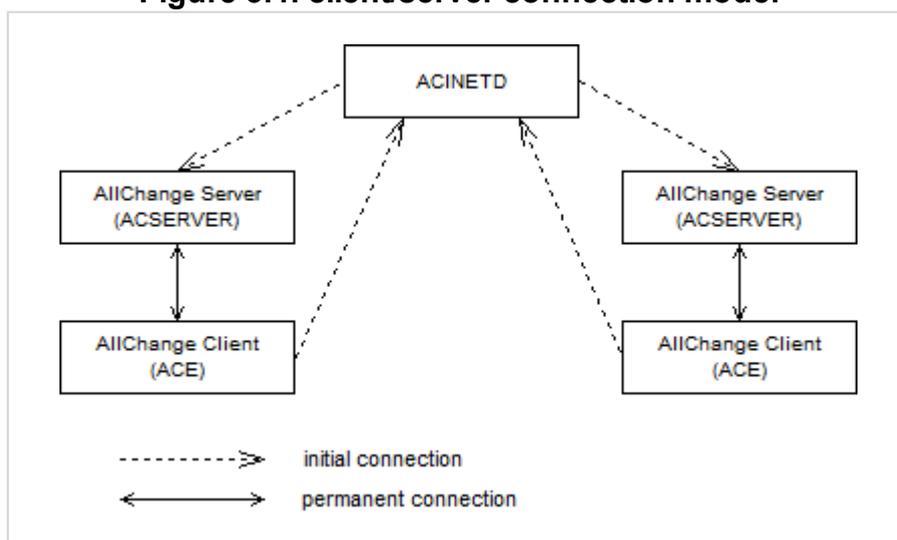
### Server Configuration

Before any user-level client **AllChange** program can be run using client/server access an **AllChange** server must be started on the server machine. The server program provided with the **AllChange** system is called ACSERVER: the executable is `acserver.exe` and resides in the `WIN32` subdirectory of the **AllChange** system directory. Each client program requires one corresponding ACSERVER to be running.

This may be accomplished by a program running on the remote machine automatically launches an `acserver` on demand, when it detects a client starting up. A service program, `ACINETD.EXE`, is supplied with **AllChange** which provides the services necessary for this.

Whenever an **AllChange** program is invoked the server's `ACINETD` will detect the connect request and start an `ACSERVER` for the client. From then on the `ACSERVER` and the client communicate with each other directly; the `ACINETD` continues to listen for new client connections. This scheme is illustrated in [Figure 3.4](#). The System Administrator will need to configure `ACINETD` to achieve this.

**Figure 3.4: client/server connection model**



When it starts up, ACSERVER needs to know the location of the **AllChange** system directory, just like any **AllChange** client. It determines this like any other **AllChange** program — environment variable `ACDIR` if it exists, otherwise from the registry. The server installation creates this registry entry.

ACSERVER ignores the client's idea of the location of the system directory. In contrast, once connected the client informs ACSERVER what project directory to use by passing it an `ACSERVERPROJDIR` variable which may be specified as part of the project definition (see Client Configuration). This scheme makes sense since clients can choose what project directory they wish to use but not what system directory is to be used.

Error messages are logged via the system's event logger; they may be found in the **Application Log** of the **Event Viewer** in **Administrative Tools**.

## Port Numbers

Clients and servers require a unique, mutually agreed *port number* at which to rendezvous. If they do not use the same port number they will not rendezvous; if they use a port number used by a different application they risk interfering with that application, and vice versa.

The `ACINETD` accepts connections on the default port numbers (7877 and 7876). If necessary, this may be changed, see later sections.

## Configuring ACInetd for Windows NT

Log on to the server; you will probably need your System Administrator to do this in order to have the permissions that will be necessary.

You will need to perform the following steps:

### Install the ACInetd

If during the server only install you installed the **ACInetd**, then skip this step. If you selected to install it later manually then from a Command Prompt:

- change directory to the directory containing the `ACINETD.EXE` (the `WIN32` subdirectory of the **AllChange** system directory)
- run `acinetd -install` (or `acinetd -reinstall` if it is already installed) to install it as a service in the Win32 registry database.
- Create a registry entry for the **ACInetd** service that you wish to use - see below

### Configure the ACInetd

Use the **Services** program in **Control Panel** to configure the **AllChange Inetd** Service to startup manually or automatically as desired. It is also possible to set the service to *log on* as a special **AllChange** account (see below).

### Set the directory permissions

The permissions for the **AllChange** project/ system/VC File/ executable directories must be modified as required depending on the user account under which the **ACInetd** service is running, see [Security and Access Issues](#).

### Start the Service

Start the service manually from **Services** or restart Windows as appropriate.

The ACSERVER executable, `ACSERVER.EXE` and the ACC executable, `ACC.EXE`, that **ACInetd** will invoke must live in the same directory as the `ACINETD.EXE` by default.

ACSERVERS invoked from `ACINETD` run with the permissions of the `ACINETD`, normally the System Account. It may be desirable to create a special **AllChange** account on the server and configure the **ACInetd** to run under this account. Remember that this account must have all required permissions to **AllChange** files on the server: read and write to the project directory, read only to the system directory. **AllChange** clients will then no longer require access to (most) **AllChange** files since all accesses will

actually be performed by ACSEVER. **AllChange** clients will still require appropriate permissions to wherever the VC files reside, depending on which client/server variant is chosen.

The foregoing should be sufficient for most sites. There are, however, some additional facilities which may be useful in certain circumstances; these may be skipped until needed.

### ACInetd Command Line Arguments

<b>-install:</b>	installs ACInetd as a service
<b>-reinstall:</b>	installs ACInetd when it is already installed (e.g. for a new version)
<b>-remove:</b>	uninstalls the ACInetd service
<b>-name &lt;service-name&gt;:</b>	specifies the name for the service - default is <b>All-Changelnetd</b> if none specified
<b>-displayname &lt;display-name&gt;:</b>	specifies the name for the service - default is <b>AllChange Inetd</b> if none specified

### ACInetd Registry Entries

Parameters for ACInetd are stored in the registry under the key:

HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\AllChangelnetd\Parameters\<servicename>

where the *servicename* is

- **acserver** - for client/server operation
- **acweb** - for browser based operation

Each service may have the following entries:

Program:

The name/path of the executable for the service. If not specified, the default is **acserver.exe** for acserver and **acc.exe** for acweb, located in the same directory as the ACINETD.EXE.

CommandLine:

Command line for the executable specifying any command line arguments. The default is no command-line arguments.

Port:

Service name/port number to listen on. Default is **acserver/7877** for acserver and **acweb/7876** for acweb. The client must be invoked with the same port to connect to the service.

KeepAlive:

Send TCP/IP **keepalives** on the socket if this is set to **True**, This should be a REG\_SZ (string) registry entry. If this is set for the acserver service then acserver may be able to detect when a connection to the client is broken and exit. This could then frees up locks which might otherwise be blocking things such as backing up.

KeepAliveTime:

Specifies the time in seconds between keepalive messages. This should be a REG\_DWORD registry entry. If this entry is not present the Windows default will be in effect, this is usually 2 hours. If setting the KeepAliveTime to very much less than 2 hours this could cause connections to be lost more frequently than if no keepalives were sent, for example 1 hour might be suitable.

Trace:

This causes it to write an entry to the Event Log noting the client IP address and spawned program process id (PID) for each accepted connection if this is set to "True". For **acserver** this is intended to help track down an ACSEVER which has continued to run when a client has gone away unexpectedly: the TCP/IP protocol does not specify when (or if) a server will detect that a client is no longer running. If the Administrator knows a client PC may have crashed in this situation he can search the event log to find the PID of the ACSEVER connected to the client's IP address; the PID can then be

used in the supplied `KILL.EXE` utility (or other application such as `PVIEW`) to kill the rogue `ACSERVER`.

Disabled:

Disable listening for connections on this socket if this is set to **True**.

Use `regedit.exe` to create and modify the required registry entries. **Caution:** obey Microsoft's standard warnings when editing the registry: you can impair or disable Windows with incorrect changes or accidental deletions; take a backup first.

### Running multiple instances of **ACInetd**

If you wish to simultaneously run two different versions of **AllChange** (e.g. whilst upgrading) then you will need to run two different versions/instances of **ACInetd**.

**ACInetd** listens for a connection on a particular port and spawns the required program (`ACSERVER` or `ACC` (located in the same directory as the `ACINETD`) whenever an **AllChange** client (e.g. `ACE` or the browser interface) tries to connect. To run a different version of a client you need the corresponding **ACInetd** to be running and listening on a different port. However, **ACInetd** is an Windows service, and Windows only allows a single instance of a service to run.

To address these problems:

1. Since Windows identifies services by their *service-name* different versions of **ACInetd** may be run simultaneously by installing them with different *service-names*. *service-display-name* is what is shown in the Services Applet: a suitable string should be chosen. Note that if an installed service is to be removed at a later date the same `-name service-name` command-line argument must be used as when it was installed (any `-displayname` is ignored).
2. A different port number should be specified in the registry for the service

So to run a new version of **AllChange** client/server alongside an existing version follow these steps:

1. Install the new **AllChange** into its own area on the server (say `c:\newacsys`), and similarly on the client
2. Enter:  

```
c:\newacsys\win32\acinetd -install -name "AllChangeInetd-newver"  
-displayname "AllChange Inetd (New version)"
```
3. Use `regedit.exe` to specify the port in the key: `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\AllChangeInetd-newver\Parameters\acserver\Port`
4. Start the service manually/automatically as usual
5. Specify the port number in the project definition

To uninstall the new **ACInetd** service, enter:

```
c:\newacsys\win32\acinetd -remove -name "AllChangeInetd-newver"
```

### Client Configuration

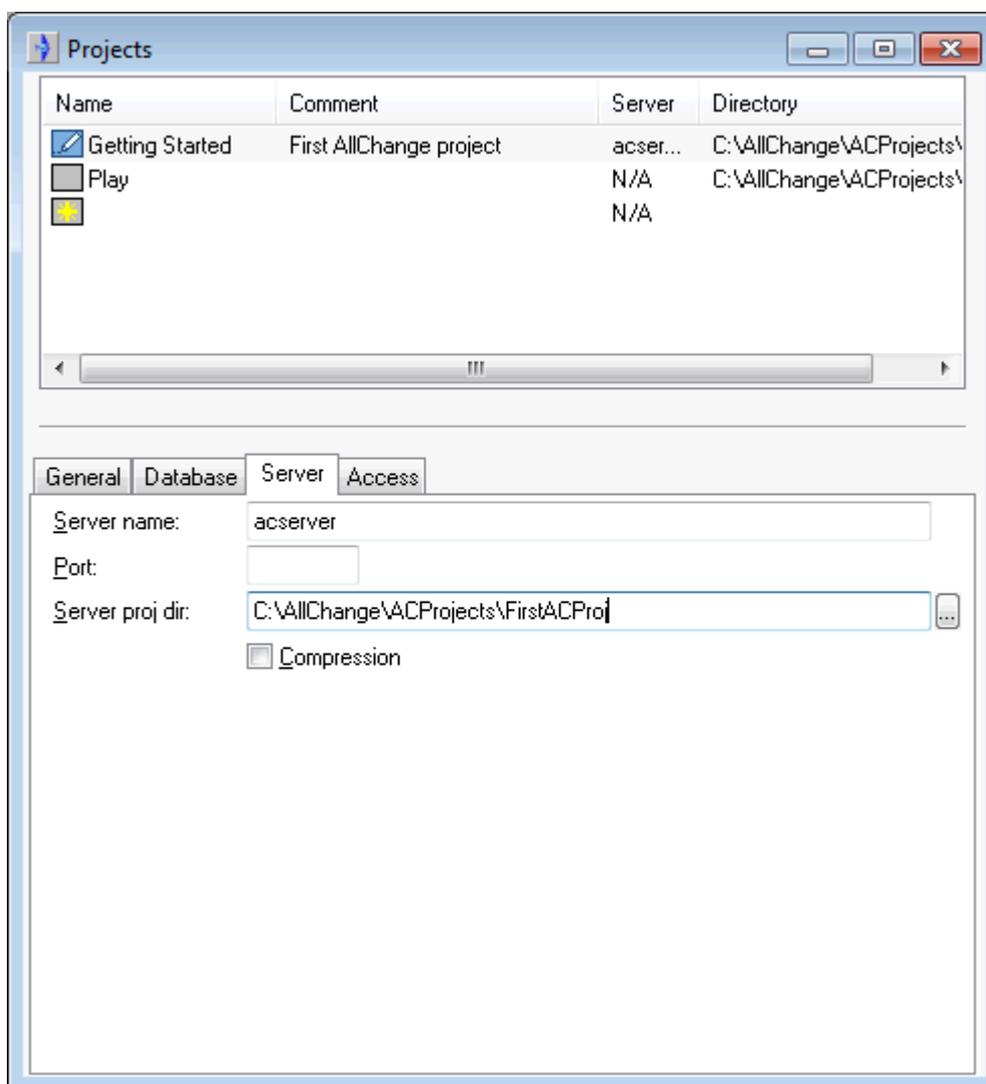
End-user **AllChange** client programs which are capable of running client/server — `ACE` and `ACC` — determine whether to run client/server or not when they start up by looking at the project definition, the environment or the registry for an `ACSERVER` entry, as explained below. Other client programs (e.g. `ACCONFIG`) *never* run client/server, regardless of this setting. If a client attempts to run client/server but the server has not been set up or is not accepting requests for some reason an error is issued and the client exits. The same executable is used for both client/server and non-client/server operation.

Client **AllChange** programs (`ACC` etc.) need three pieces of information in order to contact the **AllChange** server successfully:

1. The name of the machine running the server. This is known as `ACSERVER` and may be set either to the IP address of the server machine or, preferably, to the server machine's name — in the latter case ensure that each client knows how to map the name to the correct IP address, e.g. by editing `lmhosts`. If `ACSERVER` is not specified or is blank then the client will not run client/server.
2. The rendezvous port number. This is known as `ACPORT` and if unspecified the client will use the same behaviour as the server, i.e. it looks first for a service named `acserver`, then uses `7877`.
3. The path to the AllChange project directory *from the server's point of view* (since it is the server which needs to open configuration files). This is known as `ACSERVERPROJDIR` and must be specified.

This information may come from the project definition, environment variables, entries in the registry or from the command-line, as described in [Defining the AllChange Environment](#).

The environment variables/registry entries/project definitions entries required are `ACSERVER`, `ACPORT` and `ACSERVERPROJDIR` respectively, e.g.:

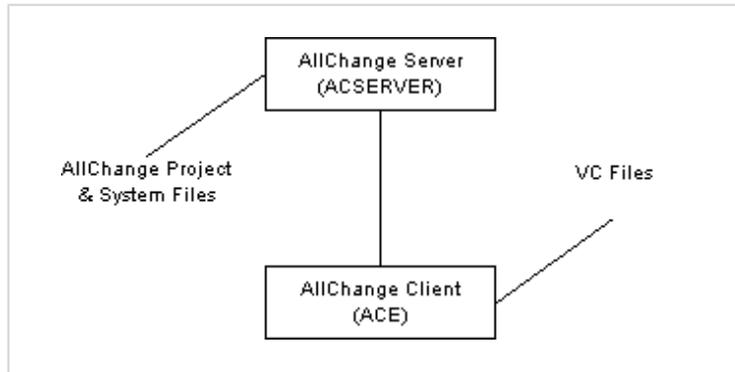


It is `ACSERVER` that determines whether an **AllChange** program attempts to run in client mode at all: if `ACSERVER` is specified and is not the empty string the program will (only) act as a client, otherwise it will (only) access the system/project directory files directly.

**Client/Server — Direct Access to VC Files**

[Figure 3.5](#) illustrates the basic client/server access model, with direct access to VC files. **AllChange** project and system files are accessed only by ACSERVER. VC files are accessed only by the client.

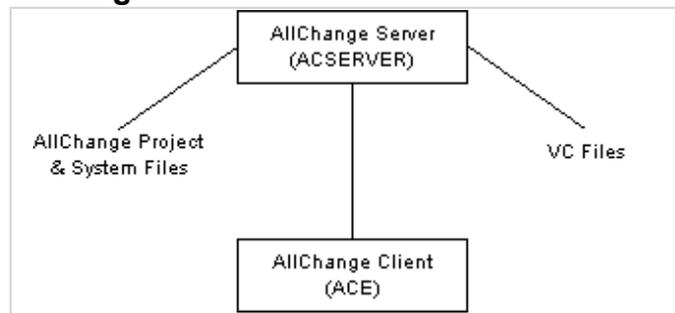
**Figure 3.5: client/server access, with direct access to VC files**



**VC client/server**

[Figure 3.6](#) illustrates the basic VC client/server access model. **AllChange** project and system files, and VC files, are accessed only by ACSERVER.

**Figure 3.6: VC client/server access**



When running client/server, the system may also be configured so that ACSERVER supplies client/server access to the VC files, as it does to the support files in the project and system directories. This provides the same efficiency and security benefits for VC files as for other **AllChange** files:

- protection of VC files against accidental or deliberate alteration/ deletion
- protection of VC files against any access (including read) other than through **AllChange** programs
- no requirement for any direct access to VC file location from clients
- considerable speed improvement for many VC file operations, especially on heavily loaded or slow network, and concomitant improvement for other applications due to reduced network traffic

VC client/server mode is activated by selecting the **VC client server** configuration option ([Plan | Configuration Options | VC tab](#) in ACCONFIG).

It may also be enabled on for an individual client by creating a `VCClientServer=True` entry in the registry (**AllChange** subkey).

VC client/server mode can only be enabled provided ACE is running client/server, otherwise it is automatically disabled. Any system that can run client/server can also run VC client/server. VC client/server mode may be introduced into an existing client/server but non-VC client/server system at any time by following these instructions, so the decision does not have to be made when the system is first set up.

Once running VC client/server, *all* operations on VC files can (and should) be performed by the ACSEVER program running on the server. Ensure that the `commands.ac` and function definition files (plus any report format files and any site-defined function files) use the ACCELfunctions dedicated to dealing with VC files (`putaway()`, `delvcfile()` etc.). These files as shipped are correctly configured for these requirements.

As described in the preceding section, ACSEVER will need to find the appropriate registry entries to tell it, among other things, where the **AllChange** system directory is located in order to successfully perform VC operations.

Whenever a VC operation involves a client workfile this will be automatically transferred to/from the server as necessary. Only the server accesses VC files; these are never transferred to/from the client. Consequently only the server needs any access to VC files, and all VC files must be accessible to the server wherever they are located (including if they are on another machine); only the client needs any access to workfiles, and all workfiles needed by a particular client must be accessible to that client wherever they are located.

Ensure all VC directories and files are fully accessible to ACSEVER and at most read accessible — or completely inaccessible if preferred — to clients. This means restricting the share holding VC files to read-only or not sharing at all; if the files reside on an NTFS partition they should be owned by a special AllChange user who has full control over them, with everyone else having only read access or no access at all to them. Run the ACSEVER as the user who owns the VC files; this means configure the ACINETD service which spawns the ACSEVER to "log on as" this account.

All new VC directories and files will then be created with the correct permissions. If upgrading from a non-VC client/server system you will want to replace all existing permissions on all directories and files. For example, they should be changed so that they are owned by AllChange, grant AllChange "Full Control", and have any permissions for "Everyone" removed or at least reduced to "Read". The Windows System Administrator may be required for this.

Since the server will now be performing all operations on VC files and directories it must be handed the correct path from its point of view. This is ultimately derived from an absolute path stored in a part's location field: probably just in /'s location field, or perhaps in top-level subsystems too. There are three possibilities for setting absolute locations, ranging from simplest to most complex:

1. [Recommended] Use the variable path `$$ACPROJVCDIR\VCFILES` on the / part. The `ACPROJVCDIR` environment variable is automatically created by the application to point to the project directory it is currently using to access VC files taking into account client/server and VC client/server. This means that the location field should not need to be modified when changing between different client/server access modes. This is the default location used when a new database is created.
2. Use UNC paths (e.g. `\\machine\share\...`). Note that in this case the ACINETD service *must* "log on as" a special account, not as the System Account (recommended anyway). Also note that the permissions on the share itself must grant full control to this special account; everyone else may be granted read-only or no access.
3. Use drive letter paths from the server's point of view (e.g. `c:\vcfiles`). Since the client should never need to access the VC files it does not matter that the path makes no sense from the client's point of view.

In addition to protecting VC files from direct client access, VC client/server operation may provide performance enhancement and reduced network traffic for VC file operations. This is a result of the fact that direct access results in the copying of entire VC file contents from/to the server, whereas VC client/server access results in the copying only of workfiles from/to the server. (It is important in this context to keep VC files on the same machine as that running the ACSEVER; if they are on another machine VC client/server may still be used, but ACSEVER will effectively have to copy them, losing any speed improvement.) When a VC file is large but the workfile is small — common when the VC file contains a fair number of historical versions which have not been calculated as a small set of differences from their predecessors, and likely

to be exacerbated over time — there can be a considerable saving in network activity. This is obviously of benefit on a slow or heavily loaded network.

The only disadvantage is that the server CPU and memory are now used for some of the work that the client would do in non-VC client/server mode. A slow server CPU, hard disk swapping or inadequate memory could lead to poorer response time for other users of the server, including **AllChange** users; if the network is fast or the VC files are *not* considerably larger than workfiles (few versions, small differences or lots of small files) it is possible that the advantages would not outweigh the drawbacks. We do not expect this to be the case in practice, but a site may wish to conduct its own performance tests. If VC operations are found to hog CPU time, try creating a `VCLowPriority=True` entry in the `AllChange` subkey of the appropriate registry entry: this lowers the priority at which VC commands are run.

## Security and Access Issues

### About Security and Access Issues

There are three aspects to security and access to the **AllChange** data:

1. Access to the **AllChange** tools
2. Access to the operating system directories where the data is stored
3. Access to the SQL database where the data is stored

All should be given due consideration to ensure that you do not permit access where it is not wanted/required.

Access to the **AllChange** tools is governed by user name together with [user registration](#), [roles](#) and [command access definitions](#). By default your user name is taken from your Windows logon user name. This is secure in most situations, however if access to the **AllChange** tools is granted over the internet, for example, additional security measures may be required. This is accommodated by configuring AllChange tools to require a logon and password, see [AllChange Login](#) for details.

Access to the operating system directories is governed by the network access permissions set for these directories. See [Directory Permissions](#) for details of **AllChange** requirements.

Access to the SQL Database is governed by the authentication method used, see [Accessing and All-Change Project Database](#), see also [SQL Server Permissions](#).

### Directory Permissions

It is important to set up the directories containing **AllChange** system files to have appropriate permissions to allow users to perform valid operations but not remove and/or amend files they are not entitled to amend.

[Figure 3.8](#) lists the requirements for end user, **AllChange** Administrator and ACSERVER (services end users when running client/server) access to files in various **AllChange** system directories in order to run AllChange.

**Figure 3.8: Required access to system files**

File	Permissions			
	User Direct	User Client	Admin	Acserver
<b>System Directory</b>				
projects.ac	Read	Read	All	None
All Others	Read	None	All	Read
<b>Executable Directory</b>				
All	Execute, Read	Execute, Read	Execute, Read	Execute
<b>Project Directory</b>				
acacts.log	Read, Write	None	All	Read, Write

All Other	Read	None	All	Read
<b>CR/Baseline Attachment Directories</b>				
All	All	All/None	All	None/All
<b>VC Directories</b>				
All	All	All/None	All	None/All

Where:

<b>User Direct</b>	means end users when not running client/server
<b>User Client</b>	means end users when running client/server
<b>Admin</b>	means administrative users
<b>Acserver</b>	means the ACSERVER program

You should use your network/ operating system to implement these access controls (see also [System Configuration](#) and [Operating System Specifics](#)). Please note that "all" access does mean read, write, delete *and* create. Note too that the permissions are the minimum required for the system to function. The system *may* be set up with less strict permissions, but if security is a concern the above should be followed. In particular, if (and only if) running client/ server end users should not have write access to files in the project directory or attachment subdirectories, and if running VC client/server they should not have write access to VC files either.

Note that this table gives the access permissions required to run the AllChange applications. For workstation installation additional access requirements may be necessary, in particular read access may be required to a number of files in the system directory.

Note the clarifications and additions below.

### *System Directory*

If running client/server the *only* file in this directory to which clients require any access is `projects.ac`. The client's `ACDIR` variable/registry entry (see [Windows Registry Settings](#)) must point to this directory.

### *Executable Directory*

Execute access is (obviously) required for any **AllChange** program. Since the default installation puts these in a `WIN32` subdirectory of the **AllChange** system directory direct access and execute permission will be required there. Clients also require read access to a few other files in this directory. If these have been installed locally for a client then no access is required to the system executable directory. The client's `ACEXEDIR` variable/ registry entry (see [Windows Registry Settings](#)) must point to this directory. During workstation installation there is an option to do this for you.

Keeping executables local to clients may give a (slightly) faster startup time than reading them from the network, but upgrading is more involved.

### *Project Directory*

At present, only `ACCONFIG` actually *create* files in the project directory; `ACSERVER` only needs update permission (though this may change in future).

### *CR/ Baseline Attachment Directories*

If using CR or baseline file attachments, these are stored under a subdirectory named `crattach` or `blattach` respectively under the project directory. If not running client/server, clients will require all access to these directories. Conversely, if running client/server, clients will require no access to these directories and `ACSERVER` will require all access.

## *VC Directories*

If not running VC client/server, clients will require all access to these directories and ACSERVER will require no access. Conversely, if running VC client/server, clients will require no access to these directories and ACSERVER will require all access.

The default is to store all VC files and directories within a subdirectory named `VCFILES` under the project directory, though it is possible to store them elsewhere.

## *Workspace Directories*

As stated in [Modes of Operation](#), workfiles in workspace directories *always* require direct access from a client and *never* from the server.

## *Pool Directories*

Clients will need read access to workfiles in pool directories in order to make any use of them. In order to write to/delete from them they will usually need direct access with appropriate permission. However, if running client/server it is possible to configure pools to be updated by ACSERVER instead — see [Pools](#).

## *Home and Windows Directories*

Files are sometimes accessed in the user's home directory or Windows directory. These must obviously be fully accessible from the client.

## **AllChange Login**

When running **AllChange** programs (e.g. ACE) client/server (only), the **AllChange** server (ACSERVER) may be configured to require a login with password from the end-user.

Normally **AllChange** does not require users to "login" to it. The user has to log onto the network initially, with a username and a password, and **AllChange** accepts the logged on username as having been authenticated.

However, if you wish to allow users to run an ACE over the Internet, connecting directly to your network server's ACSERVER program via a TCP/IP port additional security is needed.

In this situation, the user must be logged onto his local network (i.e. logged onto Windows) under a licensed **AllChange** username, but since he does not have to log onto the server network he has not been authenticated.

When ACSERVER accepts a client connection it checks to see whether a file named `acspwd.ac` exists in the **AllChange** system directory (on the server). If so the username supplied by the client must appear in this file, and if a password is associated with the name this must be supplied by the user (ACE displays a **Logon to AllChange server** dialog for this), otherwise the logon will fail. (Note that only a password is prompted for: **AllChange** takes the username from the client network/machine.) If `acspwd.ac` does not exist (as is the default case) then ACSERVER does not perform any authentication: any user may connect to it, providing the user is a [licensed AllChange user](#) or [unlicensed read-only access](#) has been enabled.

The password file is maintained from ACE/ ACC(not ACCONFIG), and they **must** be running in client/server mode. The **AllChange** command **serverpassword** is used to allow the addition, deletion and modification of login user names and passwords to the `acspwd.ac` file.

The `acspwd.ac` file contains lines of the form:

```
<username> [<password>]
```

<password> is stored encrypted; if not present on a line that user does not need a password. If necessary this file can be manipulated with a text editor (e.g. if the Administrator forgets his own password!).

If using ACSERVER authentication then whenever the Administrator adds or deleted a licensed user to/from **AllChange** (via ACCONFIG) he will also need to register/remove the user for authentication using (ACE).

As a further refinement, it is possible to tell ACSERVER to **not** require a logon for specified client IP addresses. This allows the system to be set up to allow users on the local network to get into **AllChange** without being prompted for a password, while those on other networks (e.g. over the Internet) do have to be authenticated. To do this following registry entries must be made on the server (The [AllChange Project Settings Editor](#) (or Regedit) may be used):

```
HKEY_LOCAL_MACHINE\Software\Intasoft\AllChange\<AllChange version>\Default\AllChange
HKEY_LOCAL_MACHINE\Software\Intasoft\AllChange\<AllChange ver-
sion>\Default\AllChange\TrustedIPAddresses
```

Where *AllChange version* is the version of AllChange that you are using.

TrustedIPAddresses is a string value and should contain a comma- and/or space-separated list of 4-segment IP addresses which are *trusted*. Each segment may be set to \* (asterisk) to match any address. For example:

```
127.0.0.1, 10.*.*.*, 192.168.*.*
```

might be used to allow local clients in without authentication.

To prevent any access from remote machines to ASERVER, while allowing normal access from the internal network, you could create an empty `acspwd.ac` file and set TrustedIPAddresses to just whatever is needed for your network.

### Notes:

- Server authentication is *only* available when running client/server.
- If Server authentication is employed non-licensed (free, read-only) users will have to be added into `acspwd.ac` in order to be able to log on (unless not required by their IP address).
- If running unattended scripts (e.g. `acc @file`) it is possible to supply a password on the command line by using `-EACSERVERPWD=<password>`.
- It is the existence (or not) of `acspwd.ac` in the server's **AllChange** system directory which determines whether logon is required.
- Updating `acspwd.ac` is accomplished (in ACE/ACC) by creating `acspwd.new` in the same directory, writing the new file contents to it, deleting `acspwd.ac` and renaming `acspwd.new` to `acspwd.ac`. You should never normally be left with a `acspwd.new` lying around; but if you do (e.g. a crash prior to the rename) you should restore a correct `acspwd.ac` and delete `acspwd.new` before any password updates can be performed.

## Integrations

### About Integrations

**AllChange** is integrated with various office and development tools allowing access to the **AllChange** facilities with a minimal learning curve.

This section details the installation and environment requirements to enable these integrations to be used.

### Integrated FTP support

Integrated support for FTP in **AllChange** allows the control of files on a remote machine — typically running UNIX or VMS — to be done from **AllChange** running under Windows. It is also used for deploying Web development files managed under **AllChange** — see [Web Development Support](#). FTP permits the copying of files between different machines across TCP/IP. It is typically used where no direct access is possible, e.g. the PC is *not* running PC-NFS to a UNIX server and so cannot access it directly. Any Windows client running TCP/IP with a Winsock interface is supported. The remote machine must be running an FTP server daemon, e.g. `ftpd` under UNIX. A user must have an account on a remote host in order to use FTP.

The following remote operating systems/FTP servers are supported:

- `unix` — supports most Unix FTP servers
- `VMS` — supports most VMS FTP servers
- `pc:ms-dos` — supports Microsoft PC FTP Server (supplied with Windows NT/2000/IIS) functioning with its "Directory Listing Style" set to "MS-DOS"
- `pc:ms-unix` — supports Microsoft PC FTP Server (supplied with Windows NT/2000/IIS) functioning with its "Directory Listing Style" set to "UNIX".

All VC files are maintained on the local PC network; **AllChange** and the VC tools it calls are executed only on PCs, and VC files are manipulated in the standard fashion. Workfiles, however, are accessed through FTP-aware ACCELcode, which copies them between a workspace/pool directly accessible from the local PC and the remote platform as necessary: for example, the **check out/issue** command will access a VC file on the local network to produce a workfile in some local workspace, and then it will copy the workfile to a workspace area on the remote machine via FTP. Support is also provided for using **promote** to a pool directory via FTP.

One problem which arises when using FTP is choosing between text/ascii or binary/image transfers. In the default configuration supplied, the choice of whether to perform text or binary transfers is determined by an arbitrary field named **Binary**. If this field exists then if the value is **Yes** then the transfer is binary otherwise it is text. If this arbitrary field is not defined then the operating system of the remote machine as defined in the workspace definition is used. If it is UNIX or a PC operating system, all FTP file copies are binary (rather than text) copies, as this is faster and simpler; it assumes that workfiles are not intended to be accessed on the local machine, so the distinction between text and binary can be ignored. If it is VMS, on the other hand, all FTP file copies are text as we have found that most sites only control text files and VMS FTP loses the text file identity during a binary transfer. These rules for determining the transfer mode may be tailored to individual site's requirements by modifying the ACCEL function **FTPFileType** defined in `ftpfunc.ac`.

A DLL, `FTPINT.DLL`, is provided to offer various FTP commands/transfers to and from a remote machine. This contains one exported function, `FTPCommand`, which should be called from ACCEL as:

```
call_dll('FTPINT.DLL', 'FTPCommand', user-name@host-name@host-password, command,
file1, file2)
```

It returns 0 for success or -1 for failure. Any error message is put into the ACCEL variable `dll_result_str`.

The `user-name`, `host-name`, `host-password` triplet determines which host and what user are to be used for the transfer. The DLL caches (i.e. holds open) the last connection. Whenever a new `FTPCommand` is called, if the `user-name@host-name@host-password` is identical to the previous one used then the current connection is reused; otherwise the current connection is closed and a fresh one is made. This has a huge impact on performance when transferring many files, as the FTP log in process (which would otherwise have to be repeated for each file transfer) is relatively slow. The connection is closed when **AllChange** terminates, or it may be explicitly closed using the `logoff` command. Different `user-names` or `host-names` may be used at any time, but bear in mind that the caching efficiency may be lost.

The supported values for `command` are:

**chmod:** (attempts to) set the (access) mode (specified as `file1`) of the remote file/directory specified by `file2`. FTP does not support this feature "natively". This command will only work if your site's FTP server supports the optional `SITE CHMOD` command; To determine if it does so, log in via an interactive FTP and executing `REMOTEHELP`. `mode` should be a string of whatever is required by `SITE CHMOD`, e.g. `0444` for UNIX. (Note that we have not succeeded in finding a VMS FTP which supports this and the Microsoft FTP server also does not support it). Whenever a read-only workfile is transferred to a (UNIX) FTP server this facility is used to (attempt to) set the UNIX file read-only; no error is issued if this fails.

**delete:** removes `file1` as a file on the remote machine.

**dir:** outputs an FTP listing of the remote file/directory *file1* into the (local) file *file2*. FTP defines the format of this output as "human-readable"; since it varies according to the host operating system, ACCEL code to parse this output is supplied in a user-defined function named `FTPRemoteFileList`.

**dir\_exists:** checks whether *file1* exists as a directory on the remote machine. Returns 1 if *file1* exists, else 0.

**file\_exists:** checks whether *file1* exists as a file on the remote machine. Returns 1 if *file1* exists, else 0.

**file\_exists\_msftp:** as **file\_exists**, but works for the MS PC FTP Server

**literal:** treats *file1* as a literal FTP command, passes it to the server, and sets `dll_result_str` to the (last line of) the reply returned

**get:** transfers remote file *file1* to local file *file2* as an ASCII (text) file.

**get\_bin:** transfers remote file *file1* to local file *file2* as a binary file.

**logoff:** just logs off from host, useful when finished.

**logon:** just logs on to host, useful to check password etc.

**mkdir:** creates *file1* as a directory on the remote machine.

**put:** transfers local file *file1* to remote file *file2* as an ASCII (text) file.

**put\_bin:** transfers local file *file1* to remote file *file2* as a binary file.

**rename:** renames *file1* to *file2* on the remote machine.

**rmdir:** removes *file1* as a directory on the remote machine.

**start\_log:** starts appending all FTP exchanges into the (local) file *file1*.

**stop\_log:** stops appending FTP exchanges to the log file.

*File1* and *file2* should be local or remote path specifications according to context.

As an example,

```
call_dll('FTPIINT.DLL', 'FTPCommand', 'fred@host@password',
'get_bin', '/home/fred/abc/def.c', 'c:\unixwork\abc\def.c')
```

copies as a binary file *def.c* from */home/fred/abc* on host to *c:\unixwork\abc* on the local machine. User *fred* must have a valid account for FTP transfer with password *password* on host.

Various ACCEL functions have been provided (in `ftpfunc.ac`) to perform FTP actions. The `commands.ac` file calls these functions as necessary whenever a workfile might need accessing across FTP. The first time any FTP function for a given host is called in a session the user is prompted for his password on that host; this is then reused in future calls.

To enable FTP support the Administrator will need to follow these steps:

1. Use `ACCONFIG` to define the required ftp workspaces and/or pools. Both a local (PC) directory and the required corresponding remote directory need to be specified. The remote/FTP information is specified on the **FTP** tab of the **Workspaces/Pools** window. Workspaces may be hierarchical or not as usual. The local directory specified is used as the area where workfiles will be placed prior to/after transfer to/from the remote host.
2. If usernames on the remote machine differ from PC **AllChange** usernames then the remote machine user name for each user must be specified in the user registrations (**Access | User Registration** in `ACCONFIG`) for each host to be used.
3. Ensure the necessary accounts exist on the remote host.
4. Select the required FTP enable options. In `ACCONFIG` the **Plan | Configuration Options** window, the **Misc** tab allow FTP Workspaces, Pools and Deployment to be enabled separately as required.

Whenever an activity is performed in a workspace and FTP support is enabled, **AllChange** checks whether the workspace is an ftp workspace and if it is the FTP functions are executed.

### Import from Email

It is possible for remote users to submit information to **AllChange** via Email. This allows a user who does not have direct access to a running **AllChange** to perform various operations by using an email system to construct a message containing the details required and sending it to a user on the system which does run **AllChange**. At some point **AllChange** will detect the arrival of the mail message, parse its body to retrieve the details and perform the operation with these details on behalf of the remote user. At present only Creating a CR or Casting a vote is supported.

*Note that email submission is only supported for named user licensing and not for concurrent user licensing. Only users who are registered **AllChange** users may submit via email, the email address from which email submissions are sent must be the email address specified for the registered **AllChange** user.*

Support for email submission is provided by two layers:

1. ACCELfunctions are provided in the supplied `mailfunc.ac` to cause mail messages to be read from the mailing system, the body to be parsed, and either CRs to be created containing the corresponding details or a vote to be cast.
2. DLLs are supplied to interface to the particular mailing system in use: their job is to retrieve and delete messages sent to the special user in the mail system. Interfaces are provided to access mail using MAPI via `ACMAPIEX.DLL` or using SMTP/POP3 via `SMTPLAIN.DLL`. The source code for these is provided; it may be modified if required at a site, or it may serve as a model for writing an interface to another mail system. Note that POP3 mail requires a password, which is normally prompted for. The password can be entered in `mailfunc.ac` (or can be coded to be picked up from profile etc) instead for automatic email import.

Two actions at least must be performed before using this facility.

1. If a special 'dummy' user (e.g. **AllChange**) is to be used to receive email submission then the Mail Administrator must create a mail account for the special user.
2. The **AllChange** Administrator must decide how and when the **AllChange** system is to check for the arrival of new emails, and set up to execute this. The **ImportFromEmail** function defined in `mailfunc.ac` may be used to accomplish the processing of mail messages and must be invoked periodically. This could be done manually from within ACE from time-to-time alternatively a [scheduled job](#) could be created to cause this to be done at regular intervals.

The **ImportFromEmail** function will import from the mailing system specified by the [Mail system](#) configuration option unless this is overridden by the [ImportFromEmailMailSystem](#) configuration option. It will import the from the mailbox named **AllChange** unless an alternative mail box is defined in the [ImportFromEmailMailbox](#) configuration option.

**MAPI** and **SMTP** mail systems are supported using MAPI and SMTP/POP3 respectively to process the email.

MAPI may be used to connect to Microsoft Exchange Server. It may be possible to access other mail systems through MAPI if you have the appropriate software drivers.

SMTP/POP3 is used when using an SMTP/POP3 mail system. You will be able to use SMTP/POP3 if you have installed a TCP/IP providing a Winsock interface.

If SMTP/POP3 is used then the POP3 server may be specified in the [ImportFromEmailPop3Server](#) configuration option.

It is also possible to specify the vote initiated email to be sent from the email account/mailbox which is to imported from (e.g. AllChange) instead of the email being sent from the user causing the vote to be initiated, see [SendVoteMailFromUser](#) configuration option. This then allows a voter to cast a vote by simply replying to the vote initiated email.

The import function reads each message in the mailbox — which has not been marked as read (MAPI only) — processes the command and parameters specified in the message text, and then:

- If the import has been successfully processed the message is deleted and an acknowledgement email is (normally) sent to the originator, the [ImportFromEmailReply](#) configuration option may be set to false to prevent this.
- If the message was recognised as directed at **AllChange** but either the command is not recognised or the command failed then the email is marked as read and a failure message is sent to the originator.
- If the message was not recognised as directed at **AllChange** then if the [Import-FromEmailNotForAllChange](#) configuration option is specified the action specified is taken, otherwise the email is left as unread (note that this has no effect under POP3).

In order to enable Email submission in **AllChange** you should:

- Enable the [Mail Import feature](#) in the configuration options.
- Ensure the [Mail System](#), [Mail box name](#), [Not for AC](#), [Send reply to sender](#) configuration options are appropriately set if needed
- If required add the **ImportFromEmail** function to a menu using the **ACCONFIG GUI | Menu Item** window.

The `mailfunc.ac` functions require the mail message body for the messages it processes to be in a certain format. The format chosen corresponds to that produced by (certain) Web servers when a Web browser user fills in a "form" to submit an email. If a non-Web email submission system is used it may be possible to supply a "template" for the email message which the user reads in and completes. If it is not possible to produce a message body in the format expected by the supplied `mailfunc.ac` then either the parsing code can be altered appropriately, or the message text can be post-processed before being passed to the function.

For email submission the command to execute must be encoded in the message text as a line `Remote-Command=command`. At present only `NewCR` and `Vote` is supported. Other lines in the text specify additional information required by each command in the form `field=value`:

### *NewCR Command*

Fields for the new cr to be created should be specified as required. All fields, except **class**, are optional (unless **AllChange** has been otherwise configured). *Field* is as required by the `newcr` command. If the field `AssignTo=` is present the CR is assigned to the specified user. As a special case the CR text is specified in a `CRText=` entry, and this must be the last entry in the message body. Any subsequent lines are treated as text continuation lines.

Arbitrary fields may be specified using the old name or the new assigned name, e.g. `Type=` is equivalent to `arb1=` in the default configuration.

An example message sent to **AllChange** by Fred might be:

```
To: AllChange
Subject: Anything you like
summary=A CR created from a mail message
RemoteCommand=NewCR
AssignTo=Fred
class=RemoteCR
Type=arbitrary field 1 contents
CRText=This CR has been created from a request received via email
This is the 2nd line of text.
```

The reply would be:

```
To: Fred
Subject: RE: Anything you like
CR number CR00046 has been created for you.
```

### *Vote Command*

The first line of a vote email should be the vote to be cast on a line of its own. This should be followed by lines with Database=<CR | Part | Baseline>, Item=<item voted on>. The comment and arbitrary filed values may be specified on subsequent lines. The Vote initiated email will contain all mandatory information and lines for all compulsory additional fields and so a vote may be cast by email by simply replying to this message (to the appropriate mail box) with the vote to be cast as the first line of the email and completing any mandatory fields (e.g. Comment). *Note that the sender of the vote initiated email may be set to the mailbox to which vote emails should be sent using the [SendVoteMailFromUser](#) configuration option thus allowing a vote to be cast by simply replying to the vote initiated email.*

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.

If composing the email by hand a suitable email might be:

```
Accepted

RemoteCommand=Vote
Database=CR
Item=SCR00010
Comment=My Email Submitted Vote
```

### **AllChange Web Browser Access**

**AllChange** may be accessed via a Web browser. This allows, for instance, users at a remote location to run **AllChange** even though they are not connected directly to the network.

The following topics are covered below:

[System Requirements](#)

[Installation Instructions](#)

[Authentication Methods](#)

[Registering Users](#)

[Facilities Supported by the Web Interface](#)

[Time Outs](#)

[Temporary Directories](#)

[AllChange HTML Report Files](#)

[Web Browser Interface Options](#)

[Firewalls](#)

[Dates and Times](#)

[Miscellaneous](#)

[URL Link Direct to Page](#)

## *System Requirements*

### **Server**

- Web Server capable of running Java Servlets and Web Applications (or separate HTML web server and servlet server)

- Java 2 (version 1.3.1 and upwards)
- Network capable of using TCP/IP sockets

There are 2 options to set up a web server capable of running Java servlets :

1. Use Apache Tomcat (on UNIX or Windows) which is free to use and is easy to set up. It supports web applications, which makes setting up the **AllChange** Web Interface simpler. You can either run this stand-alone or configure it to work with IIS or Apache web server.

The Apache Tomcat servlet engine is supplied on the **AllChange** CD (or may be downloaded from the Members area of [www.intasoft.net](http://www.intasoft.net)) in the `extras\webac\tomcat` directory along with the Java SE Runtime Environment in `extras\webac\JRE`. Please note that these versions are for Microsoft Windows only.

For other versions, please go to <http://tomcat.apache.org> and <http://java.sun.com> respectively. Further information can be found in the installation instructions.

2. Use another web server which is able to run servlets and web applications. You can also purchase add-on modules to existing web servers such as IIS. This is the easiest option to set up. Examples include:
  - Sun Application Server Platform Edition
  - Adobe JRun
  - IBM Websphere Application

If you already have a web server capable of running Java servlets and web applications then we would recommend that you use this. If not we recommend that you use the supplied Tomcat server.

## Clients

The **AllChange** Web Interface is accessed via a web browser, so no special client software is required. Users can access the **AllChange** Web Interface from most web browsers, including :

- Internet Explorer (version 6.0 and upwards)
- Mozilla Firefox (Version 2.0 and upwards)

The browser must support Frames, Java and Javascript.

Users will need to have the Java Runtime Environment installed on their client machine. This needs to be the Sun Java Runtime Environment. This is supplied on the AllChange CD (in `extras\webac\JRE`) or can be downloaded from the Sun website (<http://www.java.com>).

Cookies need to be enabled if you want to retain user settings.

We recommend that Internet Explorer users set up Explorer to check for new versions of a page every visit to the page. To set this up:

- Run Internet Explorer
- Select **Internet Options** from the **Tools** menu
- Go to the **General** tab
- Click on the **Settings** button under the **Temporary Internet Files** heading
- Set '**Check for newer versions of stored pages**' to '**Every visit to the page**'

## Installation Instructions

1. If using Tomcat (supplied on the **AllChange** CD) please see [Additional Notes for Apache Tomcat for Windows](#) in addition to these instructions. We suggest using the Tomcat web server if you do not already have a web server capable of running Java servlets (IIS does not support Java servlets).

2. During an **AllChange** Full Installation the Web Interface option must have been selected in order to copy the web interface file from the CD to the **AllChange** system area.
3. The AllChange Server installation must have been done on the **AllChange** server machine and the acinetd service must be running.
4. If you did not copy the web interface file to a specified location during the server installation you may need to copy the acwebapp.war file from the webac directory in the **AllChange** system directory.

The exact method of installing the **AllChange** Web Interface will depend on the Web Application Server that you are using. Some will have an automated deployment option for web applications. Check your Web Application Server documentation if you are not sure.

5. In your web server or servlet engine you will need to deploy a new web application, specifying the acwebapp.war file (the **AllChange** Web Interface) (e.g. <acsysdir>\webac\acwebapp.war) as the location. The application name and URL context path can be set as preferred. The URL context path will be /acwebapp as default.

e.g.

Application Name	URL Context Path	Location
AllChange Web Interface	/acwebapp	<acsysdir>\webac\acwebapp.war

This will set up the web application which will include all the HTML files and servlets etc. All the application settings have been set and should not need to be adjusted for the web application to run.

6. You will now have a directory in your webserver folder which contains all of the files necessary to run the **AllChange** Web Interface. This directory is structured as a **web application** as defined in the Java Servlet Specification v2.2.

The permissions for each directory inside the web application will need to be set up as shown below:

Web Application Directory Permissions	
<webappdir>\acwebapp\config	read/write
<webappdir>\acwebapp\attachments	read + AllChange User write
<webappdir>\acwebapp (& subdirs not listed)	read
<webappdir>\acwebapp\WebiLog	read/write
Web workspace directories	read/write + AllChange User read/write/delete
Temporary Directory	read/write + AllChange User read/write/delete

Where <webappdir> is your web application directory where the AllChange Web Application has been installed: (If using Tomcat this might be C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps)

The only user who will need access to the web application is the user who is running as the web server (either your main web server or the Java servlet server if running directly through the servlet server).

In addition, the user running as the AllChange service will need write access to the attachments directory.

Directories of web enabled workspaces need to have read/write permission set for both the **AllChange** user and the web server.

*Important Note* : The web interface will attempt to use localhost as the server where the **AllChange** server software is located. If this is not the case then you will need to edit the `options.txt` file (you may need to create it if it does not exist) in the web interface `config` subdirectory (e.g. `<tomcat home>\webapps\acwebapp\config`) and add the following line.

```
acserver=<servername>
```

#### 7. File Upload Software:

- a. Copy the `commons-fileupload-1.3.1-bin.zip` file from the CD (from `extras\tomcat\commons-fileupload-1.3.1-bin.zip`) to a temporary directory; unzip, and copy `commons-fileupload-1.3.1-bin\lib\commons-fileupload-1.3.1.jar` directly (i.e. no sub-folder) to the library directory of your web server. For example if using Tomcat copy to `C:\Program Files (x86)\Apache Software Foundation\Tomcat 6.0\lib`.
- b. Copy the `commons-io-2.4-bin.zip` file from the CD (from `extras\tomcat\commons-io-2.4-bin.zip`) to a temporary directory; unzip, and copy `commons-io-2.4\commons-io-2.4.jar` directly (i.e. no sub-folder) to the library directory of your web server. For example if using Tomcat copy to `C:\Program Files (x86)\Apache Software Foundation\Tomcat 6.0\lib`.

Without these files, files cannot be uploaded through the web interface.

8. Copy the `Waffle.1.5.zip` file from the CD (from `extras\tomcat\Waffle.1.5.zip`) to a temporary directory; unzip, and copy all files from the `Waffle\Bin\` directory directly (i.e. no sub-folder) to the library directory of your web server. For example if using Tomcat copy to `C:\Program Files (x86)\Apache Software Foundation\Tomcat 6.0\lib`. Without these files, logging on using Windows Authentication will not be available; if you do not wish to enable Windows Authentication logon, you do not need to copy the files. Without this file, logging on using Windows Authentication will not be available.
9. You will probably need to restart your webserver depending on the webserver you are using. If using Tomcat you will need to restart the service.
10. The **Web Browser Interface** feature needs to be enabled for the project (s) you will be running with the web interface by using the **AllChange** Configuration Editor, see [Configuration Options, Features](#). This will enable two function files, `htmlfunc.ac` and `webifunc.ac`. The Web Interface URL, and temporary directories may also be specified here, see [Temporary Directories](#).
11. The project(s) that are to be accessed by the web interface should ideally be set up as non client-server. This will improve speed and bypass the password system used by **AllChange** itself. If you have passwords enabled on an **AllChange** client/server project, then you will not be able to run this project through the web interface as standard. In order to enable this project to run, you will need to list the IP address of the machine running `acserver` as a trusted machine.
12. Before using the web interface it is necessary to determine the authentication method to be used - see [Authentication Methods](#)
13. If using AllChange authentication then [users must be registered](#).
14. If you wish to run under HTTPS see [Running on HTTPS](#)
15. If using Windows Authentication for SQL Server then the user that the `acinetd` service runs as will need access to the SQL Server database for each AllChange project, see [Authenticating the web client](#).
16. To access the **AllChange** Web Interface on the client machine, simply point the web browser to the root location of the web application. If you set up your URL context path as above then you would access the interface via :

```
http://<servername>:8080/acwebapp/ if using Tomcat
```

e.g. `http://webserver:8080/acwebapp/` (if your web server was called webserver)

`http://<servername>/acwebapp/` if not using Tomcat or Tomcat is integrated with IIS

e.g. `http://webserver/acwebapp/` (if your web server was called webserver)

### Additional Notes for Apache Tomcat for Windows

1. Run the Java SE Runtime Environment setup program (`extras\jre\jre-7u55-windows-i586.exe`) on the server where the servlet engine is to be set up, most likely the web server.
2. Once this has completed, run the Apache Tomcat setup program (`extras\tomcat\apache-tomcat-6.0.41.exe` **on the AllChange CD**) on your web server machine. Follow the instructions in the help provided with Apache Tomcat to set up Apache Tomcat.
3. To deploy the `acwebapp.war` file to Tomcat run the Tomcat manager, e.g. `http://localhost:8080/manager/html`. Select WAR file to upload in the deploy section. This will be either where you copied the war file to during Server install or the webac subdirectory of the **AllChange** system directory, e.g. `<AllChange system directory>\webac\acwebapp.war`.
4. You will need to restart the Tomcat server before using the AllChange web interface

### Authentication Methods

The **AllChange** web interface supports two methods of authentication:

1. **AllChange** authentication - users are registered as **AllChange** web interface users with a user name and password. This is in addition to the requirement to register as valid **AllChange** user. The **AllChange** web interface will present a logon screen for users to enter their credentials as defined by an **AllChange** web interface administrator - see [Registering Users](#) below
2. Windows Authentication - The users windows account is used to authenticate the user. The **AllChange** web interface will present a Windows Account Logon button and no user name or password details need to be entered, the windows logon details will be used, this allows for single sign on. In order to use Windows Authentication the user must be logged onto the domain. Internet Explorer running under Windows should pass the domain user name to **AllChange** automatically when logging on, other browsers will prompt for a domain user name and password. Windows Authentication must be enabled to permit this authentication method - see [Windows Logon](#)

Both authentication methods may be used at the same time, for example, some users may not be logged onto the domain to allow Windows authentication and so should use **AllChange** authentication, whilst other users are logged on to the domain and may use the Windows authentication.

Users logging on using Windows authentication do not need to be defined as web interface users as per **AllChange** authentication unless they wish to perform web interface administration functions (managing user names and passwords),

### Windows Logon

It is possible to log onto the **AllChange** Web Interface using your Windows user name, instead of using the **AllChange** username / password combination. This is known as Windows Authentication or Single Sign On.

If you have successfully logged on to Windows, your Windows user name will be passed to **AllChange** when pressing the Log On button next to 'Use my Windows Account'.

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.

To enable Windows authentication, the waffle files must have been copied to the web server's library directory as detailed in the [installation instructions](#). The following lines should be uncommented from the supplied **web.xml** file (found in `acwebapp\web-inf` folder):

```
<filter>
<filter-name>SecurityFilter</filter-name>
<filter-class>waffle.servlet.NegotiateSecurityFilter</filter-class>
</filter>
<filter-mapping>
<filter-name>SecurityFilter</filter-name>
<servlet-name>logon_wa</servlet-name>
</filter-mapping>
```

More information can be found at <http://dblock.github.io/waffle/>.

## Registering Users

If using **AllChange** browser interface authentication then before anyone can log on they need to be registered as browser interface users, this is in addition to the requirement to [register as AllChange users](#). Administrator facilities within the browser interface are provided to do this.

In order to register users from the Log On page enter your user name and password and then click on the **Admin** button. Note that if no users have yet been registered (e.g. first time use) you will not need to enter a user name or password

Add any users in that you want to be able to access the web interface. Each user must have a user name and a password. The user name *must* match with the corresponding **AllChange** [registered user name](#).

In addition users may be classified as **Admin** users. This is distinct from the **AllChange** registered administrative users and identifies those web browser interface users who may perform browser based administrative tasks (i.e. register users).

If there are no Admin users defined then any user can access the Admin page, it is therefore strongly recommended that at least one Admin user is allocated.

Currently there are no facilities to modify a user name or password. User names and passwords are stored in the `config` subdirectory of the `acwebapp` directory.

Users may log on using their Windows logon name if Windows Logon is enabled.

## Facilities supported by the Web Browser Interface

The browser interface supports most **AllChange** functionality as supported by the Windows Interface including:

- Support for CR, Part and Baseline operations
- Reports may be run
- Eval command is supported allowing arbitrary ACCELstatements to be evaluated.
- Set Role
- Attach Workspace
- Mailing

## Time-Outs

The ACC(**AllChange** Command Line) that is running on behalf of the web server has a time-out which is set to 30 minutes (1800 seconds) as default. This means that if you haven't used your currently running

web interface for more than 30 minutes, **AllChange** will automatically quit and you will need to log on again. To change this value, there is an optional environment variable that can be set, called `ACCTIMEOUT`. This will need to be set in the registry area for the acinetd web interface section, under **CommandLine**. E.g. `-EACCTIMEOUT=3600` will set the time-out to 60 minutes, see [ACInetd Registry Entries](#).

The web application also has a socket time-out of 5 minutes which means that if the web application does not receive a response from **AllChange** (after issuing a request) for over 5 minutes then it will give up waiting.

The web server's Java servlet engine will also have a settable time-out that will invalidate a session. This means that if you have not used the web interface in the current web browser for this amount of time, the session will become invalid and both the web interface and **AllChange** will quit.

You can adjust the socket time-out by adding a line into the options.txt file in the Web Application. See [Web Browser Interface Options](#) section for further details.

If you try to request another page from the web interface while **AllChange** is still producing output from the previous request, the web page will not be updated until that original output has finished. You will have to wait until **AllChange** is free again before a new page can be displayed.

### *Temporary Directories*

Download directories are created for downloading files from the web server to the web browser when file attachments are extracted. The default download directory will be in the attachments subdirectory of the web application directory. A subdirectory of the users name is created to store the files in. If you wish to change the root location of the downloads directory you can do so by setting a configuration option (WebInterfaceAttachmentDir) on the Features tab, using [ACCONFIG](#). Note that this must be a location that can be accessed by the web server.

A temporary directory is used to hold CR/Part/Baseline Text when it is updated and to hold copies of file attachments when they are uploaded. The normal temporary directory which is stored in the `ACCEL` variable `tmpdir` may not be appropriate for the web interface. The location of the temporary directory used by the browser interface can be changed by setting a configuration option (WebInterfaceTempDir) on the Features tab, using [ACCONFIG](#). The specified temporary directory must have read/write permissions for the **AllChange** user and the Web Server user.

### *Mail*

In order to use the **AllChange** email facilities through the web interface, the **Mail System** option in `ACCONFIG` must be set to SMTP.

### *AllChange HTML Report Files*

The HTML for Visual `ACCEL` dialogs and some prompt lists are held in a special report format file — **AllChange** HTML Report (`.ahr`). These are similar to textual report files (`.rep`). Lines beginning with `!` are evaluated, lines beginning with `#` are ignored and all other lines are treated as literal.

These files are read in by **AllChange** and the resulting output is sent to the web server. These are used mainly for modal dialogs. Use `<MODAL>` at the start of `ahr` reports to lock them. The marker `<EOM>` must be put at the end of modal output reports.

### **Editing Web Interface Report Files**

Most of the web pages generated in the web interface are produced from **AllChange** reports and the `.ahr` files mentioned above. These files are editable so that you may tailor the pages to your own specifications. The report files are all named according to the pattern `wi*.rep`.

The functions that are used by the web interface are held in the `webifunc.ac` file.

## Web Browser Interface Options

Various options can be set by modifying the `options.txt` file found in the `config` subdirectory of the web application. These options are *global* options and are described below.

`debug=false | true` — this option can be set to true or false. If not present then it will default to false.

With debug set to true, input and output from **AllChange** to/from the web server will be output to a file called `debuglog.txt` in the `WebiLog` subdirectory of the web application.

`acserver=<AllChange Server>` — this option can be set to specify the host machine name of the **AllChange** server.

This option is used to override the standard `acserver` definition of `localhost`. It needs to be set if you are running the **AllChange** server on a different machine to the web server

`acport=<port no>` — this option can be used to override the default port number used to connect to the **AllChange** `acinetd` service.

The default port that the Web Interface will connect to **AllChange** on is 7876. It will also use two further sockets on ports 7875 and 7874 (the two ports below the main one). If you wish to change this, set the main port to your chosen number. The other two sockets will be the two preceding socket port numbers. Note that you will need to set up the **AllChange** `acinetd` service to use your chosen port number first. See [Configuring ACInetd for Windows](#) on how to change the port numbers used by `acinetd`.

`SocketTimeout=<seconds>` — this option is used to override the default timeout for sockets connected to **AllChange**.

The default timeout for reading data from AllChange is 5 minutes (300 seconds). After this time, if no data has been received from AllChange then the web application will give up.

`trustStore=<location of keystore file>` — this option can be set to specify the location of the `.keystore` file used for running the web interface using a HTTPS connection. This should point to the `.keystore` file used by the web server (i.e. Tomcat) for HTTPS connections. The location of the keystore file needs to be in a directory accessible from all clients wishing to use the file transfer facility over HTTPS. For example, you might set up a share for it, e.g. `\\acserver\keystore\keystore`

## Firewalls

The **AllChange** Web Interface uses three sockets, the default port numbers of which are :

7876 - Main Socket  
7875 - HTML Output Socket  
7874 - Data Input Socket

The main socket can be altered to another port number if required (see [Web Browser Interface Options](#)). The other two port numbers will be the two numbers preceding the main port number.

If you have a firewall set up, the port numbers of the three sockets will need access through the firewall. Note that these are the socket numbers used by the **AllChange** server, the other ends of the sockets are assigned unique port numbers on the web server.

## Running on HTTPS (using SSL)

The AllChange web interface is able to run on an HTTPS server. To set up Tomcat to run using SSL on HTTPS, refer to the Tomcat documentation in your Tomcat installation or on <http://tomcat.apache.org/tomcat-6.0-doc/ssl-howto.html>.

Once you have set up your web server (i.e. Tomcat) to run using HTTPS, you can access the web interface in the normal way, using `https://` instead of `http://`, e.g. `https://acserver:8443/acwebapp` (Tomcat uses port 8443 for SSL instead of the standard 443). Everything will run as normal, except for file transfers.

To enable the file transfer facility to run in the web interface you will need to specify the location of your key-store file in the `options.txt` file in the config directory. Please see [Web Browser Interface Options](#) for more information.

The keystore file is the one that is used by your web server (i.e. Tomcat) to store the certificate to run on SSL, e.g. `C:\.keystore`. This file is accessed by the client machine and so will need to be distributed to any clients wanting to use file transfer over https. The location of the file is specified in the [options.txt](#) file is from the clients' point of view.

## Dates and Times

All dates and times displayed and entered using the web interface will be in the timezone of the server running AllChange. This timezone is indicated in the left hand (navigation) frame.

Dates can be entered and displayed in either US format (mm/dd/yyyy) or UK format (dd/mm/yyyy). This option can be changed by using the Options Dialog (from the Misc menu, Options item). This also affects times, the US format being hh:mm:ss AM/PM and the UK format using the 24 hour clock.

## Miscellaneous

- The ACCEL variable **winsys** is set to **web** when running the web interface.
- Some of the HTML files included in the web interface have substitutions in them that the Java servlet will expand. This includes :

`%%surl%%` - the URL location of the servlets

- The CR buffer size used by the Web Interface CR Browser is determined by the global variable `Global_WEBICrBufferSize` which is set to the default value of 50 in the function `WEBI-Initialise` in `webifunc.ac`

## URL Link Direct to Page

It is possible to construct a URL (web link) which will open the AllChange web browser interface at a specified page. This may be used, for example, in emails from AllChange to allow access to a CR from a link in the email. In the out-of-the-box system the email generated on assignment of a CR will include such a link to allow the CR to be viewed.

2 instructions are supported on the link:

1. **project** - The AllChange project to use. Optional, if unspecified will prompt for project
2. **autologon** - if using Windows Authentication will automatically log the user on
3. **bypasslogon** - Bypass the logon page if the user is already logged on and go straight to the requested page
4. **startpage** - the page to open the web browser on
5. **user** - the user name to fill in on the logon page

An example link might be:

```
http://-
/loca-
lhost:8080/acwebapp/portal?bypasslogon=true&project=Default&startpage=servicerequests?f
```

This link would open the web browser at the CR viewer page for CR CR00113 if the user *ben* was already logged on to the web browser interface. If not already logged on, then the logon page would be shown first with the user *ben* filled in. This is a link similar to what is generated by the system for the assign CR email link.

The `user=ben` may be omitted from the URL in which case the user will need to log on normally and the user will not default on the logon page.

The general format for the user name on the URL is:

```
user=<username>
```

The general format for the startpage on the URL is:

```
startpage=servicerequests?function=<function required><optional parameters>
```

Where the optional parameters are specified as:

```
%26<parameter>=<value>
```

Note the %26 is an encoding for the '&' character. This is necessary to ensure that the options are taken as parameters to the startpage instruction. In addition the space character must be encoded as %20 and the '/' character as %2F

The following functions are supported:

#### **crrequest:**

This opens a CR View page. Parameters are:

**CRNumber** - The CR to be displayed

**CRTab** - Which tab on the CR View to display. Possible values are:

nontab - display the last tab viewed

wicrtab1 - General tab

wicrtab4 - Text tab

wicrtab5 - Parts tab

wicrtab6 - Baseline tab

wicrtab7 - CRs tab

wicrtab8 - Attachments tab

wicrtab9 - Status log tab

wicrtab10 - Votes tab

Example: `function=crrequest%26CRNumber=CR00113%26CRTab=nontab`  
to show CR00113 with the last tab viewed

#### **partrequest:**

This opens a Part View page. Parameters are:

**PartName** - The part to view as a full path to the part. The '/' characters must be encoded as %2F

**PartTab** - Which tab on the Part View to display. Possible values are:

'nontab' will display the last tab viewed.

wipatab1 - Part tab

wipatab2 - Versions tab

wipatab3 - CRs Affecting tab

wipatab4 - CRs Solved tab

wipatab5 - Status Log tab

wipatab6 - Check-outs tab

wipatab7 - Baseline tab



**standardpage:**

This may be used with the `rename` parameter set to `wipalist` to show the part browser page. Parameters to `wipalist` may be used to set filters on the browser. `wipalist` parameters include:

**PartFilter** - Value of the Filter By drop-down list

**PartSearch** - Value of the Condition (as ACCEL)

**PartVersions** - value of the Versions drop-down list

**PartSubsystems** - Value of the Subsystems drop-down list

A value of 'same' for any parameter will use the last setting on the browser.

Example: `func-`

`tion=stan-`

`ardpage%26rename=wipalist%26PartFilter=same%26PartSearch=same%26PartVersions=same`

To show the part browser with all filters set to the last value used

**Web Development Support**

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**.

Support functions are contained in the file `webdfunc.ac`. This support consists of two elements:

- Functions are provided which can parse an HTML file looking for links. Any links found which refer to the local development area — which is the state they need to be in during the editing phase — can be replaced 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 [Integrated FTP support](#).

To enable Web deployment support the Administrator will need to:

- Select `EnableFTPDeployment` in the configuration options (**Plan | Configuration Options | Misc tab**).
- Define the substitutions to be performed on links (URLs) (`ACCONFIG Workspace| Webmap`). Any links encountered which start with the **Local URL** (e.g. `http://localhost/`) will have that changed to the **Remote URL** (e.g. `http://www.company.co.uk/`)

The supplied `Websrcclass` and `web_cyclelife-cycleare` intended to illustrate a possible life-cycle for Web source. When a version enters the `deploy` status any links in the workfile which match those described in the **Web Mappings** ([Web Mapping](#)) will be substituted. The file will then be deployed across FTP to the remote machine, to a location relative to the workspace directory defined in the workspace definitions.

**Graphical Statistics**

An independent graph drawing program (`acstats.exe`) is supplied with ACE under Windows; it is designed to draw graphs from statistical data extracted from **AllChange**. It interprets commands sent to it (via OLE) to set the graph data and style, displays the graph and allows the user to alter certain styles and copy it to the clipboard. It uses the standard Microsoft graph drawing package known as "MSChart".

We have supplied two CR reports which use this package (comments in the report files give further detail):

`crgstate.rep` — shows the current "state" of the CR system; x-axis may be chosen by the user (statuses, assignees etc.)

`crgtime.rep` — shows what has happened to CRs over the past year

To run: just report on CRs with the desired report format file as usual. When the report has successfully completed the graph will appear on the screen. Menus allow the user to alter the style of the graph and to copy the graph to the clipboard (for pasting into another application — this is what must be done to print the graph).

The supplied `.rep` files illustrate how **AllChange** communicates with `acstats.exe`; this may be used as a starting point for further reports. Here is a definition of the `acstats.exe` OLE interface.

The **ACStats** program acts as an OLE server: *ProgID* is `ACStats.Application`. The syntax of commands understood by **ACStats** is:

```
<verb> [<argument>{,<argument>}]
```

i.e. a command verb, optionally followed by a (single) space and then a comma-separated list of arguments (any spaces are treated as part of the arguments).

The commands it understands are as follows:

**NewGraph**

Clears out any current graph and data, resets all attributes. *Must* be called as the first step in setting up a graph.

**GraphTitles** *graph-title,x-axis-title,y-axis-title*

Sets the respective graph titles. X- and/ or y-axis titles may be omitted.

**GraphLabels** *x-axis-label,x-axis-label,...*

Sets the labels along the x-axis, e.g. month names.

**GraphLegends** *legend-1,legend-2,...*

Sets the legends indicating which set each colour represents. Only useful in graphs with more than one set.

**GraphStyle** *graph-type,graph-style*

Sets the overall graph type and style within the type. *Graph-type* uses the numbers understood by `mschrt20.ocx`. *Graph-style* is not used.

**GraphColours** *colour-1,colour-2,...*

Sets the colours used by each set (or by each point in a one set graph). *Colours* must be a number in the range 0 to 15, as follows:

0 Black	4 Red	8 Dark gray	12 Light red
1 Blue	5 Magenta	9 Light blue	13 Light magenta
2 Green	6 Brown	10 Light green	14 Yellow
3 Cyan	7 Light gray	11 Light cyan	15 White

**GraphSetData** *y-value,y-value,...*

Sets the y-values (data points) for one set of the graph. Each `GraphSetData` command defines a complete new set of data. A graph may be composed of one or multiple data sets. A single data set is used where you have just one set of values to be plotted, e.g. `crgstate.rep` may show the total number of CRs assigned to each individual. Multiple data sets are used where you have more than one set of values to be plotted, e.g. `crgtime.rep` shows the number of CRs over a period time

which were submitted, closed or still open: a different coloured line/ bar is used for each of the statuses.

#### ShowGraph

Displays the graph. This must be called as the last step in setting up the graph.

There should be the same number of legends as colours as data sets. There should be the same number of values in each data set, and the same number of x-axis labels as values in a set.

Once an OLE connection to **ACStats** has been established you may send it new graphs/ data as often as desired provided each new graph is started with the `NewGraph` command. You may close **ACStats** whenever you please.

#### 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.

The MS Project integration should be enabled in the [Configuration Options, Integrations tab](#). This will enable the functions for integrating with MS Project in the supplied file `mprojfunc.ac`. 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 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.

Each CR exported from **AllChange** automatically sets the following MS Project attributes:

<b>Task name</b>	to the CR summary
<b>WBS code</b>	to the CR number
<b>Priority</b>	to the CR's <b>Priority</b> field if a CR arbitrary field exists with this name.

Other fields are exported by default which may be changed as desired/ needed by setting any of the following ACCEL user-defined variables used by `mprojfunc.ac`. All of these variables may be affected via ACCONFIG: the [Integrations](#) tab on **Configuration Options** has a number of `MSPProject...` options which are used to set the corresponding ACCEL variables.

#### pjNotesFromCRText

ACCEL expression for what to set a task's Notes to. The default is the section of the CR text whose title is given by the **MSPProjectNotesFromCRText** configuration option; by default, this is set to "Description"

#### pjResourceFromCR

ACCEL expression for what to set a task's Resources to (which effectively means who is doing the task). The default is the value of the **MSPProjectResourceFromCR** configuration option, which defaults to `cr_assignee`.

#### pjDateStartStatusAttrib

The name of an attribute on a status through which a CR passes which is to be treated as the *actual* start date of the task. The default is `MSPProject_Start` (should not need changing).

#### pjDateFinishStatusAttrib

The name of an attribute on a status through which a CR passes which is to be treated as the *actual* finish date of the task. The default is `MSPProject_Finish` (should not need changing).

#### pjDateBaselineStartFields

A list of the names of CR (arbitrary) fields which are to be treated as the *anticipated* start date of the

task. Multiple fields are permitted to accommodate CRs of different classes which use different arbitrary fields. The default is the value of **MSProjectDateBaselineStartFields**, which defaults to "" (i.e. not used)

#### **pjDateBaselineStartStatusAttrib**

The name of an attribute on a status through which a CR passes which is to be treated as the *anticipated* start date of the task. The default is [MSProject Bline Start](#) (should not need changing). This is only used if **pjDateBaselineStartField** is empty.

#### **pjDateBaselineFinishFields**

A list of the names of CR (arbitrary) fields which are to be treated as the *anticipated* finish date of the task. Multiple fields are permitted to accommodate CRs of different classes which use different arbitrary fields. The default is the value of **MSProjectDateBaselineFinishFields**, which defaults to `cr_Date_Due`.

A task's actual start and finish dates – which determine the length of the bar in the Gantt chart – are set to the dates the CR passed through the statuses with attribute [MSProject Start](#) and `MSProject Finish` respectively. A task that has not yet reached its finish status is set to finish today; 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.

In addition to a bar representing the actual duration of a task, MS Project can also display a bar representing the anticipated duration of a task (which they refer 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. If you maintain a CR (arbitrary) field which holds a task's anticipated start date you should set the [MSProjectDateBaselineStartFields](#) configuration option to the name of this field; alternatively you may set [MSProject Bline Start](#) attribute statuses through which a CR passes which are to be treated as the task's anticipated start date.

A task with no anticipated start date has this set to the actual start date. Similarly, if you maintain a CR (arbitrary) field which holds a task's anticipated finish date you should set the [MSProjectDateBaselineFinishFields](#) configuration option to the name of this field; there is no alternative of a status since that would not make sense. A task with no anticipated finish date has this set to the actual finish date. Note that if no anticipated date information is held in CRs you should set all of these variables to "", in which case the Gantt chart will show only the actual duration bars with no baseline bars for comparison.

Thus in the default configuration for SCRs (CRs of class SCR) it can be seen that a task's actual duration stretches from status `InWork` to `Complete` and its anticipated duration stretches from status `Accepted` to the date stored in the arbitrary field `cr_Date_Due`. These are suitable for use with the supplied SCR life-cycle and arbitrary field definitions, but may be changed to reflect a site's tailoring of these. 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!

#### **About Integration with Mail**

An ACCEL function called `SendMail` is supplied in the `utility.ac` file. This function is called, for example, when a CR is assigned to a user, when a monitor is triggered, when a vote is initiated, when a predefined mail action is used in a life-cycle status etc. You can place additional calls if you wish users to be mailed in additional circumstances. The mail function may be called using the ACCEL `call` function as follows:

```
call (SendMail, to, cc, bcc, subject, message, force-mail_to-self)
```

The `to`, `cc` and `bcc` may be either a list of recipients or **AllChange** groups. For those mail systems which accept multiple recipients for a mail message (MAPI, Lotus/ cc:Mail, SMTP) will send just one message with multiple recipients rather than separate mails.

Any users who do not have a **Mail name** specified in the [user registration](#) will not be mailed.

The function itself may be tailored to your specific requirements or a new mail system integration added if it does not support your mailing system.

The supplied function provides an interface to the following mail systems:

- MAPI
- [LotusMail](#)
- SMTP

If you are using one of these mailing systems then you can enable mailing in **AllChange** by simply selecting the appropriate mail system in the **Configuration Options**, see [Configuration Options](#).

Although this will enable the mailing facility, it is also necessary to specify the mail name used by the mailing system for each user before **AllChange** will be able to mail that user. This is specified in the **User Registration** information for each user, see [Licensing Users](#).

When **AllChange** sends a mail, the user is prompted with a dialog containing the mail text, so that the text, subject CC or BCC recipients may be modified. This prompting may be turned off by setting the Configuration option "**EditableMailMessages**" to **False**.

If you are using a different mailing system from those listed above then the function needs configuring. ACCEL provides several different mechanisms for integrating with other tools, and any of these may be used as appropriate to your mailing system.

- operating system command line — if using this interface the mailing system will need to be able to accept command line parameters for the user(s) to mail, the subject and the message.
- DLL — this is the interface used by MAPI. This may involve you in writing some C code and recompiling the `ACINTDLL.C` file supplied (the supplied code includes the necessary code for MAPI, see [call\\_dll\(\)](#)).

If using SMTP or MAPI mail then the mail may be sent as either plain text or HTML. If the configuration option [Format emails as HTML](#) is set then by default emails will be sent as HTML. If the option is not set then emails may still be sent as HTML by explicitly prefixing the Subject with `<HTML>`, e.g.

```
call(SendMail, user, '', '', '<HTML>Subject', 'Message', false);
```

Email text containing HTML tags should be wrapped in a `<BODY>` tag, and optionally an `<HTML>` tag. Text without these tags is converted to HTML-safe text.

Many of the out-of-the-box emails sent by **AllChange** will contain HTML code if the option is enabled. This includes, for example, mails sent on assigning a CR, or by initiating a vote.

If the [Editable Mail Messages](#) configuration option is enabled, and the mail is to be sent as HTML, then the dialog used for editing the mail will use an HTML editor. This is a simple 'WYSIWYG' editor allowing basic editing of the HTML text.

Support for HTML edit controls in [Visual ACCEL](#) is available. In addition various functions in `utility.ac` may be used assist with building HTML text for emails. These are:

<b>FormatItemTextAsHtml</b>	Takes cr, part or baseline text and makes it HTML safe, whilst also formatting the text sections in bold text.
<b>MakeHyperlink</b>	Takes a URL and link text, and build the HTML anchor tag for including in an HTML email.
<b>MakeACInterfaceURL</b>	Takes database and an item and returns the AllChange URL for this
<b>MakeWebInterfaceURL</b>	Takes database and an item and returns the AllChange Web Interface URL for this
<b>GenerateURL</b>	Generates an AllChange URL for the current project on a specified item - see <a href="#">URL Protocol</a>
<b>GenerateURLForSelectedItem</b>	Generates a correctly formatted URL from the currently viewed item, or the selected item in a list - see <a href="#">URL Protocol</a>
<b>HTML</b>	Converts a text string to HTML

**MultiLineTextToHtml**                      Converts a multi line text string to HTML

### *Lotus Mail/ cc:Mail Integration*

The integration supplied with **AllChange** for Lotus/ cc:Mail is implemented using `VIMINT.DLL` which is supplied with **AllChange** in the **AllChange** executables directory. `VIMINT.DLL` needs to find the Lotus DLL `VIM32.DLL`. This will usually have been installed in the same area as the Lotus mail executables.

To allow `VIMINT.DLL` to find `VIM32.DLL` one of the following will be required:

- `VIM32.DLL` must be in the Windows or Windows System directory
- The directory containing `VIM32.DLL` must be on the `PATH`
- `VIM32.DLL` must be copied to the same directory as `VIMINT.DLL`
- `VIMINT.DLL` must be copied to the same directory as `VIM32.DLL`, and the line in `utility.ac` which reads `call_dll('VIMINT.DLL', ...)` must be altered to include the new path to `VIMINT.DLL`

In all cases, if `VIM32.DLL` itself needs other Lotus DLLs these must be copied with `VIM32.DLL`.

You may also want to put the following line in `ac.ini`: `call_dll('VIMINT.DLL', Lock-DLLInMemory, '', '', '', '')`; This will allow the last connection to the Lotus mail system to be held open, which may improve speed for sending second and subsequent mails in an **AllChange** session. The system has been configured to prompt the user for his Lotus mail password the first time (only) **AllChange** wants to send a mail.

## Development Tool Integrations

### *About Development Tool Integrations*

**AllChange** provides facilities to conform to the Microsoft Common Source Code Control Interface (MCSCCI). This means that it can be integrated with any compliant environment. This includes Microsoft Visual Basic (Version 4 onward) and Visual C++ (Version 4) and Visual Studio (including Visual C++) Version 5 development environments, Microsoft Access 97 (onward), Telelogic Rhapsody 3.0 and Artisan Real Time Studio among others. This will allow the "checking in" and "checking out" of files with corresponding **AllChange** parts from within the tool GUI. Note, however, that the interface specification is defined by Microsoft, not Intasoft; it can be used to drive **AllChange** in a simple fashion but does not provide access to some of the more advanced features. See the **AllChange** User Manual.

The **Microsoft source code control interface** option must be selected when performing an **AllChange** workstation install in order to enable the interface facilities. Note that to successfully install this integration on an NT workstation the user performing the installation will probably require Administrator privileges. This will make the changes to the Windows registry which are required.

(Note that for certain versions of VB the Visual SourceSafe server *and* client must be installed in order for the **AllChange** Source Code Control Add-In to be installed.)

The integration should be enabled in the [Configuration Options](#). It is implemented via `acscci.dll` together with `ACCELfunctions` implemented in `sccifunc.ac` and report format files `scci *.rep`. These may be tailored to alter how the integration works.

`acscci.dll` buffers information retrieved from **AllChange** and only updates this information on a **Get**, **CheckOut** or **CheckIn**. You should not therefore attempt to perform operations in **AllChange** 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 workspace/project directory(s) used by the IDE must be defined as **AllChange** Workspaces if they are to be used with **AllChange**.

The interface will attach to an already running ACE if there is one, otherwise it will start a new ACE.

Files which are under *source code control* are always available in the appropriate workspace directories and are (by default) **checked out read only** in **AllChange** unless they have been specifically checked out for edit purposes.

If it is preferred that the files for a MCSCCI project are not maintained checked out read only when not checked out for edit, then the variable **scci\_GetWithoutIssue** in `sccifunc.ac` may be set to `true` and the files will be *got* without registering in the check-outs database.

By default **CheckOut** will not prompt for which version but will check-out the default always since this corresponds to usual requirements. The variable **scci\_CheckOutPromptVersion** in `sccifunc.ac` may be altered to enable prompting if desired.

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.

## Registry Entries

The following registry entries will be made when performing a workstation install which selects the **SCCI** interface option:

Required registry entries under Windows 95/98 and NT:

```
HKEY_LOCAL_MACHINE\Software\SourceCodeControlProvider
    ProviderRegKey=Software\Intasoft\AllChange
HKEY_LOCAL_MACHINE\Software\SourceCodeControlProvider\InstalledSCCProviders
    Intasoft AllChange=Software\Intasoft\AllChange
HKEY_LOCAL_MACHINE\Software\Intasoft\AllChange
    SCCServerPath=<full path of dll>
    SCCServerName=Intasoft AllChange
```

## SCCI Environment

The following environment variables or entries in the registry may be made which will affect the behaviour of the interface. Note that these entries must be made in the `Default` project subkey in the registry regardless of whether a different subkey is usually used with ACE (since the integration has no way of knowing this). Registry entries may be modified using the **AllChange** project settings editor available from the **Start Menu**

Entries should be added to a `SCCI` subkey in the `Default` project subkey.

### **ACECMDLINE=command line**

Specifies the full path and command line for the SCCI to execute ACE. Add the `-EACPROJECT=` command line argument to specify the **AllChange** project directory if required; this is how an ACE may be run which does not use the default project. If no value is given then `ace.exe`, with no command line arguments, will be assumed. If no value or a relative path is given then your `ACEXEDIR` must be set correctly.

**ACEMINIMISE=<TRUE | FALSE>**

This determines whether the SCCI should run ACE minimised. The default value is `TRUE`. If ACE is run minimised then the `-minimise` option is passed to ACE preventing user interaction.

**ACECMDTIMEOUT=*seconds***

Specifies the maximum number of seconds for the SCCI to wait for ACE to start up. If this entry does not appear then the default value is 30 seconds.

**PUSHTOFRONT=*value***

Specifies when to upfront the IDE after a DDE conversation:

**ALWAYS** upfront after every DDE conversation.

**DEFAULT** after every error in ACE and calls to **GetCommandOptions**

**NEVER** upfront application

`DEFAULT` is the default value if there is no entry. This is only important if **AllChange** is not minimised.

## Uninstalling

In order to uninstall the SCCI integration registry entries need to be deleted.

This is performed automatically when integration with VB/VC++ is selected for removal during a **Workstation only Uninstall** (available by running setup in the **AllChange** system directory from a workstation). This is the normal way of uninstalling VB/VC++ integration. It is documented here in case the information is needed for some reason.

Registry entries to be deleted under Windows 95/98 and NT:

```
HKEY_LOCAL_MACHINE\Software\SourceCodeControlProvider
  ProviderRegKey=Software\Intasoft\AllChange
HKEY_LOCAL_MACHINE\Software\SourceCodeControlProvider\InstalledSCCProviders
  Intasoft AllChange=Software\Intasoft\AllChange
```

## Integration with Eclipse

The integration with the Eclipse IDE allows files to be checked in and out of AllChange directly from the Eclipse interface. It also supports Eclipse Team Synchronisation facility and the moving and deleting of files.

The integration is implemented as a plugin (`com.allchange.eclipse_9.0.0.jar`) together with a supporting DLL (`aceclipse.dll`) together with some ACCEL functions in `eclipsefunc.ac`.

To install the interface it should be selected during Full install to place the required files in the **AllChange** system directory.

The integration must then be enabled in the [Configuration Options](#).

The workstation install may then select the Eclipse interface to install it. This will:

- Copy the `com.allchange.eclipse_9.0.0.jar` file from the `ACINT\ACECLIPSE` directory in the AllChange system directory into your Eclipse plug-ins directory
- Copy the `aceclipse.dll` file into your `<Windows Directory>\System32` folder

If replacing an older version of the **AllChange** plug-in, start Eclipse with the `-clean` option to clean out the plug-in cache when first running Eclipse with the new plug-in version. To do this, just put `-clean` on to the Eclipse command line or shortcut, e.g. `c:\eclipse -clean`.

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. See Integration with Eclipse in the User Manual for more details.

### Integration with Araxis Merge

The integration with Araxis Merge allows the Araxis comparison and merge facilities to be used in place of the built in facilities supplied with AllChange ( VisDiffs and VisMerge). The Integration provides more powerful comparison facilities than supported by the built in tools in particular side by side file/part comparison and merge capabilities, interactive meta-baseline comparison facilities and subsystem comparison facilities.

**AllChange** uses a virtual file system facility within Araxis to provide baseline and subsystem comparison in a fashion similar to folder comparison.

In order to use Araxis Merge as your baseline/subsystem comparison tool you need to:

- Enable the [Araxis Merge Integration](#) in the **AllChange** configuration options using Acconfig, this will enable the araxfunc.ac function definition file.
- add the following to the vfspplugins.ini file in your Araxis installation area if this is not already present:

```
[acvfs]
progid=Merge.ACDFS
description=AllChange VFS
autoregister=ACDFSPlugin.dll
```

- The compare.exe application must be installed to a path which has a folder with the word **Araxis** contained in it. The default installation path is acceptable
- Each user that wishes to use Araxis Merge must have it installed and must select the Araxis compare.exe as their **Differencing Application** in the **AllChangeOptions**

Facilities supported by the integration with Araxis Merge include:

- File/Part Version Comparison
- Baseline Comparison
- File/Part Version Merge
- Folder Comparison

For full details of Araxis Merge capabilities see the Araxis User Guide.

### Integration with Windows Explorer

The integration with Windows Explorer allows various **AllChange** functionality which affects files to be available directly from Explorer.

The integration supports both 32 bit and 64 bit versions of Windows Explorer. It 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 enable Explorer integration it must be selected during **AllChange** workstation install. Note that to successfully install this integration on an NT workstation the user performing the installation will probably require Administrator privileges.

**AllChange** functionality is available from an **AllChange** context menu when you right click - see the Checking In and Out from Explorer section of the **AllChange** User Manual and the Windows Explorer Interface section of the **AllChange** User Manual.

An important consideration is that **AllChange** must be able to find a workspace for the current user corresponding to the selected files (CheckIn) or directory (CheckOut). So you may only check in from or out to a defined workspace directory, not just anywhere.

If ACE is not currently running it will be started (minimised) to service the requests. The user cannot interact with this ACE; run a separate ACE if you want full **AllChange** interaction for other purposes. The ACE will continue to run minimised to service future requests; it may be exited explicitly by right-clicking on the icon and selecting **Close**. The parameters for running the servicing ACE are the same as those which may be placed in the `SCCI` subkey of the registry, see *SCCI Environment*; these are sought first in a `ShellExtSCCI` subkey, then `SCCI` if not found. The only one you are likely to need to set is `ACECMDLINE`; you will need to set this only if you need to run the ACE with a `-EACPROJECT=...` command-line argument.

This integration is implemented via a shell extension DLL (`acshlext.dll`) which communicates with **AllChange** using OLE. In addition the ACCEL functions needed to support this facility are contained in `shexfunc.ac`.

In order to enable the Explorer Integration the `acshlext.dll` file must be found in `Windows\system32` directory. The Workstation installation will install this for you.

As supplied, **CheckIn** and **CheckOut** invoke the **AllChange Check In** and **Check Out** commands respectively. `shexfunc.ac` could be altered to modify the standard behaviour.

## Integration with Microsoft Word/Excel

### *About Integration with Microsoft Word/Excel*

The integration with MS Word and MS Excel provides facilities for:

- Version control including checking files in and out of **AllChange** directly from Word/Excel GUI
- Stamping Word documents/Excel workbooks with **AllChange** information
- Running reports directly from the Word GUI
- Creation of CRs from a requirements document in Word

This is implemented via various macros which communicate with **AllChange** via OLE. In addition the ACCEL functions needed to support these facilities are contained in `wordfunc.ac`.

In order to enable the integration the **Word Interface** must be selected during a **Full** install and then must be installed on each client for which it is required using a **Workstation** install. The workstation installation will examine the Windows registry to determine which version of Word/Excel is in use and various information about the Word and Excel installation and configuration. The following information is required:

#### **Word Program directory**

The directory containing the WinWord.EXE

#### **Excel Program directory**

The directory containing the Excel.EXE

If the installation program does not have permission to save files into these directories it will also require the following information:

#### **Startup directory**

The startup directory defined by the Word **Tools | Options | File Locations**

#### **User template directory**

The user templates directory defined by the Word **Tools | Options | File Locations**

#### **Add-ins directory**

The Excel directory containing Excel add-ins. The default is: `C:\Program Files\Microsoft Office\Office\Library`

If these registry entries are incorrect then **AllChange** will not be able to correctly install the Word interface.

The installation will request whatever information it needs and display default values obtained from the registry if these are available.

It will attempt to install as follows:

- ACSTARTUP.DOT, ALLCHANG.DOT and ACREQ.DOT into <Word-Program-Dir>\Startup
- ALLCHANG.WIZ into <Word-Program-Dir>\AllChange
- ALLCHANG.XLA into <Excel-Program-Dir>\Library

thus installing for all users of the machine. When a user first runs Word after installing the templates, they will be prompted for whether to trust the publisher of the ACSTARTUP.DOT, ALLCHANG.DOT and ACREQ.DOT files. This is because although Word will by default load templates from the <Word-Program-Dir>\Startup directory, it is not by default a trusted location. Once the user has enabled the macros in ACSTARTUP.DOT, its AutoExec macro will copy the ALLCHANG.WIZ from the <Word-Program-Dir>\AllChange into their user template directory (e.g. <app-data>\Microsoft\Word\Template), which is by default a trusted location. Any old versions of ALLCHANG.DOT and ACREQ.DOT found in the user's templates area will be removed, so that Word will use the new versions as installed.

The directory into which the ALLCHANG.XLA is installed is by default a trusted location, so the user will not be prompted to enable the macros unless the location is removed from the trusted locations. If they have already trusted the "Intasoft" publisher then they will not be prompted.

If the install was unable to install for all users it will install for the specific user as follows:

- ACSTARTUP.DOT, ALLCHANG.DOT and ACREQ.DOT into the user's startup directory
- ALLCHANG.WIZ into the User template directory
- ALLCHANG.XLA into the user's Excel Add-ins directory

## Version Control Facilities

Macros are supplied to allow various version control facilities to be available from within Word and Excel (communicating with **AllChange** via OLE), for example the Checking In and Out of documents.

To enable this facility for MS Word the supplied ALLCHANG.DOT file must reside in the Word startup directory so that it is loaded as a global template. The Workstation installation will ensure that this is present for you. Once enabled an **AllChange** menu will have been added to the Word menu bar and a toolbar has been added.

To enable this facility for MS Excel the supplied ALLCHANG.XLA file must reside in the Excel add-ins directory. The Workstation installation will ensure that this is present for you. In addition the **AllChange Integration** add-in must be enabled used **Tools | Add-Ins...** Once enabled an **AllChange** menu will have been added to the **Tools** menu

Macros have been provided to allow Check Out, Check In, Version Information, List Versions, Compare Versions (Word Only), Insert **AllChange** Field as described in Checking in and out of Word/Excel section of the **AllChange** User Manual, the MS Word/Excel Interface section of the **AllChange** User Manual and the Stamping Word/Excel Documents section of the **AllChange** User Manual.

An extra macro has also been provided on the **AllChange** menu called **Exit AllChange**. This has been provided as a means of quitting a minimised **AllChange** if Word/Excel should fail to do so on exit, or if you wish to quit the **AllChange** before exiting Word/Excel.

The Word/Excel integration will by default try to connect to an already running **AllChange** offering itself as an OLE Server with a progID of AllChange.Application, if not found a new instance of **AllChange** will be created. This behaviour can be overridden by specifying an alternative **AllChange** progID for Word to use in the registry. The key to specify is ACEOLESERVER=... and is sought first in a WordSCCI subkey of the Defaultproject subkey, then SCCI if not found, see [Windows Registry Settings](#) for details of the registry. The registry may be edited using the **AllChange** project settings editor available from the Windows **Start** menu.

In all cases if ACE is not currently running it will be started (minimised) to service the requests. The user cannot interact with this ACE; run a separate ACE if you want full **AllChange** interaction for other purposes. The parameters for running the servicing ACE are the same as those which may be placed in the SCCI subkey in the Default project subkey in the registry, see SCCI Environment; these are sought first in a Word-

SCCI subkey, then in SCCI if not found. The only one you are likely to need to set is ACECMDLINE; you will need to set this only if you need to run the ACE with a `-EACPROJECT= . . .` command-line argument.

## Stamping Word/Excel Documents

The facility to stamp word/excel documents with information from the parts database is included as a part of the word integration facilities, see the **AllChange** User Manual for details of how to use this facility.

Word/Excel document stamping is not enabled by default; you must enable the option on the **VC** tab of the **Plan | Configuration Options** dialog in ACCONFIG. Do not enable it if you do not intend to embed **AllChange** keywords in your documents/workbooks; although it does no harm to check in documents which do not contain **AllChange** keywords when automatic stamping is enabled it does waste time. Once automatic document stamping is enabled:

- Word/Excel documents can only be checked in from workstations which can run Word/Excel as Word/Excel will be run during check-in in order to perform the stamping;
- Word/Excel documents may be checked out from any workstation as Word/Excel is not required/used during check out.

The custom properties which may be used are determined by the contents of the output from the **AllChange** report format file named `pawdstmp.rep`. If you run the report in ACE on a component version ( **Part | Report** dialog; you will need to type `pawdstmp` as the **Format File**, as it is not presented in the list of format files by default) you will see lines such as:

*Status=Value of status field for part*

If you create a custom property whose name is **AC** followed by the word to the left of the = (i.e. **ACStatus** in this case) its value will be set to whatever appears to the right of the = in the report output. You may create custom properties on behalf of any such lines. We have tried to export all the information you are likely to want stamped into a document; should you wish to add any property we have not anticipated, simply edit `pawdstmp.rep` to output a suitable line: you may then attach a corresponding custom property to a document in the usual fashion. Note that arbitrary fields are exported with their Display Name, so for example the Description field (arb1) in the out-of-the-box configuration will be output as:

*Description=Value of description arbitrary field for part*

There are a few things you cannot do in document stamping:

- A document cannot be stamped by trying to run the **StampDoc** macro on a file for which an **AllChange** component has not yet been created.
- When doing automatic document stamping on **CheckIn**, the information stamped corresponds to the version while any actions associated with **Return** from edit are performed; if a field of the version is updated later on (e.g. an arbitrary field) this information will not yet be available when the stamping takes place. Of course, the **StampDoc** macro may be run explicitly when the version is later checked out to capture any new field values.
- Since stamping a document changes its contents a document will never be identical to its predecessor, even if no other changes have been made. This means the default **AllChange** behaviour of not bothering to create a new version for a file which is being returned from edit but has not actually been altered will never be activated: a new version will always be created when a stamped document is checked back in from edit. However, regardless of document stamping, we suspect that this never works for Word/Excel documents as Word/Excel often updates them even if the user does nothing to them.

## Running Reports from MS Word

It is possible to run **AllChange** reports from within Microsoft Word; the output is imported into Word with appropriate formatting. This may then be printed from within Word, included in documentation, etc.

A Word wizard, `allchang.wiz`, is supplied. This allows Word to communicate with **AllChange**, get it to run a report, and then read the output into Word. The user creates a report by selecting the **Allchang.wiz**

icon from the **General** or **Report** tab of **File | New...** inside Word. It will start ACE if it is not already running, but for speed it is preferable to have ACE already running. The reports offered are those with a **Category** of Word Wizard – see [Report Formats](#).

To enable this support Word must be able to find `allchang.wiz`. `allchang.wiz` must be found in Word's "templates" directory. The Workstation installation will copy these files from the **AllChange**system directory into the appropriate Word directory for you.

It is possible to write further report format files or alter those supplied, see [Creating and Modifying Report Formats](#).

### Create CR from Requirements Document

Word macros are supplied to 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.

To enable this facility the supplied `acreq.dot` file must reside in the Word startup directory so that it is loaded as a global template. The Workstation installation will ensure that this is present for you.

Once enabled an **Add Requirements** option will have been added to the **AllChange** menu on the Word menu bar.

To import the requirements from the current open word document, select the document section of interest, or move the cursor to the point from which to start and select **AllChange | Add Requirements**, see **AllChange** User Manual for details of the facilities available.

You may need to create appropriate classes of CR with the appropriate naming convention and arbitrary fields defined which are to be used for the import.

### Integration with Dreamweaver

The integration with Adobe Dreamweaver development environment allows files manipulated by Dreamweaver to be checked into and out of **AllChange** from within the dreamweaver environment.

The interface has been implemented using Dreamweaver's source control integration API (SCS). It is implemented via a Dreamweaver Source Control extension DLL (`acdwsacs.dll`) which communicates with **AllChange** using OLE. In addition the ACCELfunctions needed to support this facility are contained in `dmwvfunc.ac`

To enable the interface:

1. Copy `acdwsacs.dll` from the **AllChange** executable directory (`<system directory >\WIN32`) into Dreamweaver's SourceControl directory.

For Dreamweaver 4 and MX this a directory named "Configuration\SourceControl" under the Dreamweaver installation directory, (e.g. "C:\Program Files\Macromedia\Dreamweaver MX\Configuration\SourceControl").

For Dreamweaver MX 2004 this is Program Files\Common Files\Macromedia\2004\SourceControl, on the Windows drive.

It should also be noted that Dreamweaver's MX and MX 2004 both require that the dll's name is in lower-case, i.e. `acdwsacs.dll`.

For Dreamweaver 8 this is Program Files\Common Files\Macromedia\8\SourceControl

For Dreamweaver CS3 this is Program Files\Adobe\Adobe Dreamweaver CS3\configuration\SourceControl

For Dreamweaver CS4 this is Program Files\Adobe\Adobe Dreamweaver CS4\configuration\SourceControl

This will allow the user to select "AllChange" in Dreamweaver's Site Definition "**Remote Info**" page as the source control system for a site.

2. Enable the [Adobe Dreamweaver Integration](#) configuration option, this will enable the `dmwvfunc.ac` function definition file.

Each Dreamweaver site that is to be used with **AllChange** must have a workspace defined which corresponds to the site's **Local Root Folder**

The **DreamweaverSCS** key in the **AllChange** user area of the registry is used to hold dreamweaver user specific information.

See Checking In and Out from Dreamweaver in the User Manual for additional details.

### Integration with HP TestDirector

Facilities are provided to allow **AllChange** to control the versions 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. TestDirector is a Web application, using a web browser front-end (e.g. Internet Explorer) and operating in client/server mode against TD's **OTAServer.exe**. It is **OTAServer.exe** which then interacts with **AllChange** (specifically ACC) on behalf of the client, passing it requests and receiving responses. **OTAServer** runs on the machine acting as the TestDirector server; ACC should also run on that machine, using a client/server connection to ACSEVER if the **AllChange** database is actually on a different machine. It is important to understand that **AllChange** has no knowledge of, or connection to, the web browser client with which the TestDirector user interacts; indeed, this applies even if running the web browser client on the same machine as the TD server. Partly as a consequence of the foregoing, it is a requirement that **AllChange** have absolutely no interaction with the user. There are also some setup/permission issues that must be borne in mind, since the ACC is actually running on the TD server under the account running the **OTAServer.exe**, although in some sense it "appears" to be running on behalf of the user logged in to TD.

There are 2 files comprising the AllChange TestDirector interface:

- **WIN32\TDVCCI.DLL**: the DLL called by TD (**OTAServer.exe**) to issue commands to AllChange (ACC)
- **TDVCFUNC.AC**: ACCEL function file implementing the functionality

### To install

1. Install TestDirector. It is recommended that TestDirector be run as an NT service, as described in the chapter *Setting TestDirector as a Service* in the *TestDirector Installation Guide (TD\_Install.pdf)*. It is also important to follow the instructions in the chapter *Configuring IIS and TestDirector* (in the same guide), adjusting the account permissions as directed. You should select a specific user (the "TDUser") to run all the TestDirector programs/services where there is a choice. It is imagined that this is the configuration which would be used regardless of AllChange. In the examples below it is assumed that the TestDirector "Domains Repository" directory (where TestDirector files are stored, including end-user workfiles) is **C:\TD\_Dir**.
2. Install AllChange. This should be performed on the same machine as TestDirector was installed. You need to perform a "Server only" installation (since AllChange will be acting like a server, in a similar fashion to when it is servicing its own web browser clients); when asked, you do not need to setup **ACINETD** (unless this machine is genuinely to act as the AllChange server to AllChange clients). You must previously have performed an AllChange full installation; this could have been on the same machine, or it could have been on another machine, in which case the ACC will need to run client/server to the AllChange server. It is not necessary to do a workstation install of AllChange on

the client, though you may do so if you wish to run **ACE** from that workstation independently of the TestDirector interface. In the examples below it is assumed that the AllChange system directory is **C:\ACSYS**.

3. From a Command Prompt, run command:

```
regsvr32 c:\acsys\win32\tdvcci.dll
```

This will cause TestDirector (**OTAServer.exe**) to recognise and connect to AllChange as its "Version Control" server.

### To use

Before the integration can be used a few configuration steps are required for AllChange. Most of these are performed in ACCONFIG.

1. Add a class named **TestDirector** for type **component**.
2. Define a field **TDLabel** for a version arbitrary field (suggest **pa\_varb10**), class **TestDirector**, data type **String**, entry type **Read-only**, help string **TestDirector version label**. Ensure you have the **Comment** (on versions) and **Description** (on parts) standard arbitrary fields.
3. Enable the HP Testdirector Integration in the [Configuration Options Integrations](#) in Acconfig, this will enable the function definition file `tdvcfunc.ac`.
4. Ensure you have created AllChange users for each TD user. These *must* be spelt the same way as in the TD "login". It is not possible to use an AllChange username that is different from the TD login name.
5. Decide on the top-level subsystem for each TD project (e.g. **/TestDirector/TestProj**).
6. Create a suitable workspace for each intended user of each TestDirector project. As supplied, the code in `TDVCFUNC.AC` generates the name as: **TD\_DBName\_username** where *DBName* is the database name and *username* is the user's AllChange username (in all lower case), e.g. **TD\_TestProj\_admin**. This gives a separate workspace for each user of each TD project, which is what TD expects. The `ACCEL` code for this may be changed if desired. The workspace part must be that decided in the preceding step, e.g. **/TestDirector/TestProj**. The work directory *must* be that which TD will use for checkouts by this user of this project. This will be: the TD project directory (in SiteAdmin, shown as the value for **Project** within **Directories**), followed by **checkouts\username**, e.g. **C:\TD\_Dir\Default\TestProj\checkouts\admin**.
7. In **ACE**, create the top-level subsystems for each TD project.

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. HP have a "Version Control Demo" movie, which shows how version control works in TestDirector. The file is **TD75VCDemo.zip**; at present it is not publicly available, though it might be obtainable from their support. In its absence, 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.

### Advanced Notes

These notes may be useful in understanding how the integration works and troubleshooting. When all is well, they should be unnecessary. Unfortunately, when there is a problem **AllChange** generates a detailed error message for TD to use, but often it only displays a very generic message (e.g. "Failed to open VCS

database") and the **AllChange** message is lost. As a consequence the debugging support may be useful when initially configuring **AllChange** to work with TestDirector.

- **cmdfunc.ac:**  
 Changed **get\_vc\_flags** so that components of class **TestDirector** always create new version on putaway (required by TD).
- **Registry:**  
 These are registry entries used by **TDVCCI.DLL**, not ACC.  
**NOTE:** uses **HKLM**, not **HKCU**; default project is **Default**. Uses **TDVCCI** subkey. **ACPRJSET** may be used to access these entries.
  - **LogMessageTo:** Determines where TDVCCI error/warning messages are sent to:
    - **SysLog:** NT Event Log (default if not specified)
    - **Discard:** discard
    - **MessageBox:** message box on screen (use only for debugging)
    - **DBWin:** send to debug window [**DBWin32**] (use only for debugging)
    - *any other:* path to log file (must be accessible from **OTAServer.exe**)
  - **InfoCacheLevel:** Determines how much information caching **TDVCCI.DLL** ↔ **AllChange** can do:
    - **0:** never (for debugging)
    - **1:** as per SourceSafe
    - **2:** as per **AllChange** (default if not specified)
    - **3:** always (may not work properly)
  - **AllowReuseAllChange:** If set to **True** allows an existing **AllChange** to be used to serve TD, and does not quit **AllChange** on TD release/log out. Use only for debugging.
  - **UseACE:** If set to **True** expects **ACE** (instead of ACC) to be used to serve TD, and does not "log in" to **AllChange** (no username passed from TD). Use only for debugging.
- **To run ACC as Automation server:**
  - **Key:** HKEY\_CLASSES\_ROOT\\AllChange.ACC.Application\\CLSID  
**Value:** {04ACC9E0-8922-11D2-8284-00001B38A356}
  - **Key:** HKEY\_CLASSES\_ROOT\\CLSID\\{04ACC9E8-8922-11D2-8284-0000-1B38A356}\\LocalServer32  
**Value:** c:\acsys\win32\acc.exe -EAUTOMATION=Exclusive -EACOLEPROGID=AllChange.ACC.Application -EACOLESERVER=1 -EACOLESERVERLOGIN=1
  - Explanation of command-line arguments to ACC:
    - **-EAUTOMATION=Exclusive:** run from Automation; **Exclusive** means act as server *only* to this client
    - **-EACOLEPROGID=AllChange.ACC.Application:** default progid expected by TDVCCI.DLL
    - **-EACOLESERVER=1:** instruct ACC (ignored by **ACE**) to act as OLE Automation server *only*; also causes it to use **HKLM** instead of **HKCU** (default project is **Default**).
    - **-EACOLESERVERLOGIN=1:** instruct ACC (ignored by **ACE**) to require initial "logon" sequence
  - Above entries could be changed to alter behaviour if required. For example, suppose a special **AllChange** project **TDProject** has been set up for all TD projects. To get ACC to use this (instead of **Default**), append **-EACPROJECT=TDProject** to the **acc.exe** command-line in the registry. (As with other integrations, it is only possible to use one **AllChange** project at any one time.)

## Integration with Telelogic Rhapsody

The integration with Telelogic's Rhapsody design tool allows Rhapsody components to be checked in and out of **AllChange** from menu options in the Rhapsody environment without having to interact with the **AllChange** GUI.

For Rhapsody Version 3.0 the MCSCCI is supported and may be used, see [Development Tool Integrations](#). For earlier versions of Rhapsody a proprietary interface is supported as documented below.

### *Integration for Rhapsody versions prior to 3.0*

The facilities provided implement what Rhapsody states it requires in the documentation. Note that Rhapsody requires very limited facilities from a CM provider, and absolutely no interaction (i.e. no **AllChange** dialogs); consequently, **AllChange** operations have to be simple: no classes, no life-cycles, no CRs, no arbitrary fields, no baselines are used for controlling Rhapsody files, see **AllChange** User Manual for details of the facilities the interface provides.

The functions for integrating with Rhapsody are in the supplied file `rhapfunc.ac`, which could be tailored to provide more functionality if Rhapsody allowed it. It is implemented using OLE to communicate between the tools. **AllChange** must have been installed with a **Full** or **Workstation** install (otherwise OLE will not work).

In order to install the Rhapsody interface:

1. Ensure `acole.exe` is in your **AllChange** executable directory — the **AllChange** installation will place it in the `WIN32` subdirectory of your **AllChange** system directory. This is also the default executable directory.
2. Ensure `rhapfunc.ac` is in your **AllChangesystem** directory . The **AllChange** installation will have installed it here for you.
3. Copy `site.prp` from the `Acint\Rhapsody` subdirectory of your **AllChange** system directory to `Share\Properties` subdirectory of your Rhapsody installation directory (e.g. `E:\Rhapsody\Share\Properties` if it is installed in `E:\Rhapsody`). If the `site.prp` already exists you may have to merge this with the one supplied with **AllChange**.
4. Copy `ACHeader.txt` from the `Acint\Rhapsody` subdirectory of your **AllChange** system directory to the `Share\CM` subdirectory of the Rhapsody installation directory
5. Edit `site.prp`; inside Metaclass **AllChange**, change Property `AcOlePath` to have the appropriate path to `acole.exe` (e.g. `c:\acsys\win32`).
6. Using `ACCONFIG`, define workspaces for use with Rhapsody. Ensure **Hierarchical** is set.
7. Rhapsody will want the user to be able to use the **AllChange newversion** command: use **Access | Command Access** in `ACCONFIG` to allow **developer** permission for the **newversion** command, which requires **dbadmin** permission by default.
8. Enable the `rhapfunc.ac` function definition file in `Acconfig`.
9. In `ACE` ensure that the subsystems referred to by any **AllChange** workspaces exist and if not create them.

You are now ready to use the Rhapsody interface to **AllChange**.

Before you can make use of the integrated CM facilities within Rhapsody for any particular project you should ensure that the **Project Properties** have a **Subject** of **Configuration Management** and a **Meta Class** of **AllChange**. You may then check files in and using **File | Configuration Items**.

## Integration with Telelogic DOORS

The integration with Telelogic DOORS requirements management system allows DOORS Modules to be checked into and out of **AllChange** from within the DOORS or ACE interface while also permitting the creation of CRs directly from DOORS requirements.

The functions for integrating with DOORS are in the supplied file `doorfunc.ac`, together with various DOORS DXL scripts. It is implemented using OLE to communicate between the tools. **AllChange** must have been installed with a **Full** or **Workstation** install (otherwise OLE will not work).

Installation of the DOORS interface involves the following:

1. Set up of **AllChange** project to be used with DOORS. This should be performed by the **AllChange** administrator
2. DOORS user set up. This should be performed by the DOORS administrator
3. Set up of DOORS Client. This should be performed by the client PC administrator.

### Set up in AllChange

In order to use the DOORS interface with AllChange the following must be performed by the **AllChange** administrator

In order to install the DOORS interface:

1. Ensure `acdoors.exe` is in your **AllChange** executable directory — the **AllChange** installation will place it in the `WIN32` subdirectory of your **AllChange** system directory (this is the default executable directory) or in your local executable directory if this was selected at workstation installation time.
2. Ensure `doorfunc.ac` is in your **AllChange** system directory . The **AllChange** installation will have put it there for you.
3. Enable the [Telelogic DOORS integration](#), this will enable the `doorfunc.ac` function definition file.
4. Create a subsystem for all DOORS projects that are to be version controlled in **AllChange** (e.g. `/DOORS`). Each DOORS project will be represented by a subsystem within this (e.g. `/DOORS/Project1`). This will be used with a workspace, see below.
5. An **AllChange** workspace must be defined for use when checking DOORS modules in and out of **AllChange**. By default a name of `DOORSWS` should be used. If another name for the workspace is used this must be specified in the **DOORSWorkspace Configuration Option**. This is used for all DOORS work and should be associated with a subsystem which is to be used for *all* your DOORS projects as created above. This workspace should be **Hierarchical**.
6. A part class must be defined which is to be used for any DOORS modules which are checked into **AllChange**, this should have the DOORS Integration attribute. The out-of-the-box configuration includes a class called `Doors` which is suitable. See **Plan | Classes** in `ACCONFIG` to for information on the classes which are defined.
7. A CR class must be defined which is to be used for any CRs created during synchronisation, this should have the DOORS Integration attribute. The out-of-the-box configuration includes a class called `DoorsRequirement` which is suitable. The following fields must be defined for the class
  - Project (maps to the DOORS project that the object is in)
  - Module (maps to the DOORS module that the object is in)
  - ReqNumber (maps to the DOORS number that of the object (e.g. 4.1))

(The [Standard Configuration](#) has a class of CRs, `DoorsRequirement`, defined for this purpose together with the field definitions.)

It is possible to synchronise all of the requirement objects in the module in one go by selecting **Synchronise Module with CRs** from the **AllChange** menu.

### DOORS User Set up

In order to use the **AllChange** DOORS interface the user permissions in DOORS shown below are required. This should be set up by the DOORS administrator.

1. Any DOORS user wishing to check in modules to **AllChange** will need permission to archive in DOORS. This can be set up from the DOORS User Manager

2. Any DOORS user wishing to check out modules from **AllChange** will need permission to delete a module in DOORS. This can be set up from the DOORS User Manager

## DOORS Client Set up

In order to use the **AllChange** DOORS interface each DOORS client needs to perform the following:

1. Run the **AllChange** `acdoors.exe` program with the `/RegServer` option. This needs to be performed for each user/workstation that wishes to make use of the DOORS interface. You can do this by using the Windows **Start** Menu, select **Run...**, then select the `acdoors.exe` program and add `/RegServer` after the end of the filename. Alternatively you can run the **Command Prompt**, navigate to the **AllChange** executable directory and type, `acdoors /RegServer`. This will setup an entry in the registry, which DOORS will access to find out where to run it from.
2. Copy the sub directory named `Acint\DOORS\Allchang` in your system directory (e.g. `c:\acsys\acint\doors\allchang`) into your the `lib\dxl\addins` subdirectory in your DOORS installation directory.
3. Append the contents of `addins.idx` found in the `Acint\DOORS` subdirectory of the **AllChange** system directory to your current `addins.idx` in the `lib\dxl\addins` sub directory of the DOORS installation directory.
4. Append the contents of `acmenu.dxl` found in the `Acint\DOORS` subdirectory of the **AllChange** system directory to the file `project.dxl` in your `lib\dxl\config` sub directory of the DOORS installation directory.

Having configured **AllChange** and DOORS for use with the DOORS integration a new menu will have been added to your DOORS Project menu and to the DOORS Module menu called **AllChange**. This allows modules to be checked in and out of **AllChange** from the DOORS GUI and synchronisation with CRs. DOORS modules may also be checked in and out via ACE.

Modules are checked out using the DOORS module archiving facilities with steps taken to ensure that information about links is not lost.

For details of the facilities available see **AllChange** User Manual.

## File Comparison Tools

**AllChange** is supplied with a file comparison (or differencing) tool called **VisDiffs**, however some users may prefer to use a different personal favourite tool. Integration with third party tools is supported by allowing the user to specify the full path name to the file comparison tool of their choice. This is specified in the **Misc | Options** from within ACE.

One example might be WinMerge which provides side by side differencing. This is freely available, and can be downloaded from: <http://winmerge.sourceforge.net/>.

Perform the installation (installing wherever desired, this does not have to be the same as where **AllChange** is installed), then set the **Differencing Application** option in ACE to its path; if you chose the default location this will be:

```
C:\Program Files\Thingamahoochie Software\WinMerge\WinMerge.exe
```

Note that this product is not supported by Intasoft.

Another example is Araxis Merge which provides side by side comparison and merge facilities. **AllChange** supports additional comparison and merging facilities when using Araxis, see [Integration with Araxis Merge for details](#).

## Integration with SolidWorks

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. The following functionality is supported:

- Check In/Out
- Status progression
- Voting
- Synchronisation of SolidWorks Properties with AllChange arbitrary fields

The Integration supports SolidWorks 2010 SP0 or later.

The integration is implemented as an Add-in to SolidWorks which communicates with **AllChange** via OLE. In addition the accel functions needed to support the integration are contained in `sldworks-func.ac`.

To enable the integration:

1. **Install the SolidWorks Add-in**

- During a workstation install select the SolidWorks Integration, this will install the Add-in.
- In SolidWorks the **AllChange** Add-in should be enabled. Select **Tools | Add-ins**, mark the **AllChange** Add-in as active, and to activate on start up if required.

2. **Set-up in AllChange**

- Enable the [SolidWorks Integration](#) configuration option, this will enable the `sldworks-func.ac` file.
- Ensure that there are classes defined which are suitable for use with the SolidWorks integration, these should have the **SolidWorks Document** or **SolidWorks Drawing** attributes as appropriate and the **Keep checked out** attribute. The classes `SolidWorksDoc` and `SolidWorksDrawing` are defined in the [Intasoft Standard configuration](#). The class should have a life-cycle suitable for approval of SolidWorks parts and assemblies, note that voting is supported by the integration.
- Define any version arbitrary fields which are to be synchronised with SolidWorks properties. These fields should have the [field attribute](#) **SolidWorks Synchronise** set to the value required for the synchronisation. The fields `Reviewer`, `Revision` and `Part No` are defined for the `SolidWorksDoc` and `SolidWorksDrawing` classes in the [Intasoft Standard Configuration](#)
- Any fields to be shown in the SolidWorks task pane should have the **SolidWorks Show in Taskpane** attribute.
- Workspaces will need to be defined which correspond to the directories containing the SolidWorks documents for each user that is to use the integration.

For details of the facilities available see the [AllChange User Manual](#).

## Windows Registry Settings

**AllChange** uses the Windows registry to store various settings for the **AllChange** installation and for each **AllChange** project.

In most cases the registry does not need to be modified or examined and registry entries will be automatically created as required. However, there may be times when the registry needs to be modified manually. The **AllChange** project settings editor, `ACPRJSET.EXE`, may be used for this purpose. This can be run from the **AllChange** executable directory; a shortcut is created in the Intasoft Program Group during an Administrator install (only, as we do not want to encourage end users to use this tool). This will give you access to the **AllChange** settings only.

The root path for **AllChange** user registry settings is:

```
HKEY_CURRENT_USER\Software\Intasoft\AllChange\<version>
```

where *<version>* is the **AllChange** build version, e.g. 6.0.

The root path for machine registry settings is:

```
HKEY_LOCAL_MACHINE\Software\Intasoft\AllChange\<version>
```

The machine registry settings are used both for the client machines and for the server machine when running client/server (settings for the **AllChange** server). They are also used for ACC when it is either running as an **AllChange** webserver or as a dedicated OLE Automation server (-EACOLESERVER=1 command line argument).

The user registry settings are used to store personal configuration and status information for a user on a client machine.

For each **AllChange** project the user uses, a subkey will be created below the root path, in which all user settings for that project will be stored. In addition there is an entry in the machine registry root called CurVer which records the current **AllChange** version which is in use as the default. This will be used to determine which version is used by interfaces to **AllChange** such as the Word and Explorer interfaces.

**AllChange** programs decide which subkey area of the registry to use by the following priorities:

1. If an environment variable named ACPROJECT is defined its value names the subkey to use.
2. If the registry value: DefaultProject exists in the root registry entry (and is not empty) its contents name the subkey to use.
3. The default subkey to use is named Default.

Each project subkey in the registry will comprise:

- Further subkeys as detailed below
- Within a subkey entries have a *name* and a *value*; *name* may be one of the variable names detailed below; *value* may be any appropriate value for the variable; *name* is case-independent.

The registry entries may be created by the installation program, by ACEor ACCONFIG, or by the supplied registry editor. In general you should not need to modify the registry yourself.

The common environment variable names/registry entries currently recognised by **AllChange** programs, the root of the registry, the subkey relative to:

<Root>\Software\Intasoft\AllChange\<version>

they must appear in if used in the registry, and a brief explanation of the use of their value are listed in [Figure 3.9](#).

<Root> is either HKEY\_CURRENT\_USER (HKCU) or HKEY\_LOCAL\_MACHINE (HKLM)

**Figure 3.9: AllChange environment variables**

Name	Root	Subkey	Value
ACDDESERVER	HKCU	<project>	The service name that <b>AllChange</b> should offer itself as when acting as a DDE Server.
ACDDETOPIC	HKCU	<project>	The topic that <b>AllChange</b> should offer when acting as a DDE Server.
ACDIR	HKLM		<b>AllChanges</b> system directory containing support files used by <b>AllChange</b> system.
ACEXEDIR	HKLM		Directory containing <b>AllChange</b> executable programs. If not defined then the same directory as the application is run from is used by <b>AllChange</b> tools to call one another directly. This is usually the win32 directory beneath the <a href="#">system directory</a> .  The directory in which AllChange has been installed
ACHELPDIR	HKLM		Directory containing <b>AllChange</b> on-line help files. If not defined then the help files are sought in ACEXEDIR.
ACHOME	HKCU	<project>\C-common	Directory private to user used for storing private <b>AllChange</b> files (e.g. personal ac.ini or .rep files) in place of the users real home directory as specified by HOME.

ACPROJDIR	HKCU	<project>\AllChange	<b>AllChange</b> project directory containing certain configuration files and the <b>AllChange</b> database used by <b>AllChange</b> system and tailored to a particular project's requirements.
DEBUGAC	HKCU	<project>\AllChange	If set to <code>True</code> starts ACE with various debug information switched on.
HOME	HKCU	<project>\Common	Directory private to user used for storing private initialisation files (e.g. <code>personal.ac.ini</code> ). If not defined then the NT <code>HOMESHARE</code> , <code>HOMEDRIVE</code> and <code>HOMEPATH</code> environment variables, if they exist are used.
TMP	HKCU	<project>\Common	Directory for holding temporary file. If no temporary directory is specified in the registry or the environment then the Windows directory will be used.
TEMP	HKCU	<project>\Common	Directory for holding temporary file. If no temporary directory is specified in the registry or the environment then the Windows directory will be used.
VISDIFFS	HKCU		The full path to the differencing application that is to be used. If not present the <b>AllChange VisDiffs</b> will be used
OleTimeout	HKCU	<project>\AceSettings	Specifies the timeout in milliseconds when AllChange is acting as an OLE Automation client. Default value is 30000.

In addition to the environment variables and registry settings shown in [Figure 3.9](#) there are subkeys of the registry which may contain tool-specific settings: these values are not valid as environment variables. Some of the additional subkeys are:

**Acconfig:** This may contain certain settings specific to ACCONFIG e.g. the editor that you use

**AcconfigSettings:** which are applicable only to ACCONFIG such as window sizes and positions.

**AceSettings:** This may contain settings which are applicable only to ACE such as window sizes and positions, startup workspace etc.

**tool/ToolbarSettings:** Each of these sections contains the toolbar specific information for that *tool*, e.g. `AceToolbarSettings`.

## Licensing

**AllChange** supports 2 different user licensing models:

1. **Named user licensing.**  
An  $n$  user licence entitles up to  $n$  users to use the system.
2. **Concurrent user licensing.**  
An  $n$  user license entitles up to  $n$  instances of the system to be run at the same time.

For both these models normally a user's name *must* be known to the system as a registered (licensed) user before the system will allow that user access to any **AllChange** program. In addition to these users read-only access may be permitted to any number of users.

There are 3 different types of user:

1. Full Users — these have full access to the **AllChange** facilities. With an  $n$  user licence with  $m$  CR only users, then no more than  $n-m$  users may be full **AllChange** users.
2. CR Only Users — these are permitted to create and modify change requests and monitors, and have read-only access to other parts of the database. CR Only licenses are supplied at a significantly reduced cost compared to the *full users*.
3. Read-only Users — these have read-only access to the database and are unlicensed users

The title bar in ACEwill indicate which type of access the user has.

All CR only and Full users must have their user name registered to the system as a licensed user before the system will allow write access for that user.

It is the **AllChange** administrator's responsibility to define which type of access each user should have.

The initial full installation gave you the opportunity to register yourself as an **AllChange** user. Further details about a user (such as Full or CR only access) may be made and new users registered using **ACCONFIG** [User Registrations](#).

Permission for read-only access is granted (or denied) at a site on a per-project basis. By default it is disabled, as it may not be desirable for sensitive data to be visible to non-licensed users.

To enable read-only unlicensed access for a project, in **ACCONFIG** [Plan | Configuration Options...](#), **Misc** tab, select **Allow unlicensed users**.

If unlicensed access is permitted then an end-user who has not been registered (i.e. the **AllChange** Administrator has not added his name via **ACCONFIG** [User Registrations](#)) may still have **AllChange** installed on a workstation. He will be able to run end-user programs, including ACE and ACEDIT and the **AllChange** Web Browser interface, but not administrative programs such as **ACCONFIG**. Upon entering ace and attaching to a project the user will receive a message informing him that he is unlicensed and will be granted read-only access; the title bar will indicate this too. If the project has unlicensed user access disabled an error will be issued and the user will be logged off.

CR-only and unlicensed (or read-only) users are limited to a subset of **AllChange** commands. Specifically, CR-only users may only perform operations which modify the CR or Monitors databases; unlicensed users may not perform any operation which alters the **AllChange** database or VC files, nor are the Build facilities available. Both read-only and CR-only users may perform operations which view or extract data, provided they do not alter the data in any way. They may perform operations which alter workfiles (provided they have the necessary operating system permission) but will not, for example, be able to check changed files back into **AllChange**.

The following notes apply:

- Read-only and CR Only users may not use **Check In** or **Check Out** as these alter the Check-outs database, even if read-only. They may, however, use **Get Version to Directory** to extract read-only copies of versions to an arbitrary directory as this does not log the check-out.
- Read-only users may not be granted roles.
- Since no information is held in **AllChange** about unlicensed (read-only) users, facilities which require information such as the ability to send mail or use FTP will not work.
- Unlicensed users will still require read access to the SQL server database if using trusted connections.
- If running with direct access to the project directory (i.e. non-client/server) unlicensed users will still require read **and** write access to the database log file (acacts.log), just like a licensed end-user, even though they will not actually be allowed to update the database. If running client/server this will not be required as the server accesses the log file on the client's behalf. Similarly, direct access to VC files will require read access if versions are to be extracted, unless running VC client/server.
- In all cases access to files etc. will still require whatever permissions the operating system demands.

The **AllChange** Administrator may use **ACCONFIG**, [GUI | Menu Conditions...](#) to alter what menu items will be available to CR only and unlicensed users.

Intasoft reserves the right to alter/remove the facilities permitted to unlicensed users at any time. Support contracts do not cover unlicensed users, and support cannot be given to them.

## AllChange Projects

### About AllChange Projects

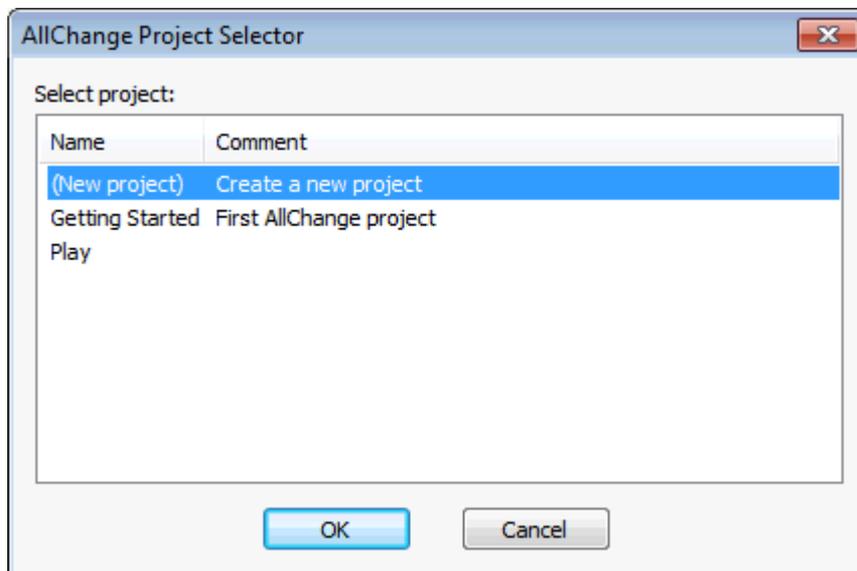
**AllChange** uses the term *project* to refer to an **AllChange** configuration and its data. The configuration comprises the definition of the CM processes etc. in use for that **AllChange** project; this information is stored in various *configuration files* which exist in the project/ system directory. The configuration files are created and defined using ACCONFIG. The data associated with the project is stored in the database and log files, together with the version control, attachment and any archive files. All of this information except for the database is usually stored in the **AllChange** project directory. Where the database is stored is determined by your SQL server installation/implementation.

**AllChange** allows you to have multiple **AllChange** projects, although we usually recommend that you have just one primary project for the live configuration management system. It is not possible to share data between projects and so careful consideration should be given to the reasons for creating separate projects.

It is recommended, for most sites, that 3 **AllChange** projects be created:

1. **Live Project** this should be the main CM project containing all the data for the site
2. **Play Project** this should be a *playground* or test area used for changing the project configuration without interfering with the live system
3. **AC4AC** this should be a version control project for the configuration of the live/play projects. This should be a very simple project with the minimum of configuration and should be used to version control all the configuration files that are tailored to site requirements for the other projects, see [AC4AC Project](#).

In general when an ACE or ACCONFIG start, if there is more than one project for the user, they are prompted to select the project to use or create a new project (ACCONFIG only).



For further details of project selection see [Using AllChange Projects](#)

Projects definitions are created administered and configured using ACCONFIG. The project currently being configured in ACCONFIG is shown in the title bar. If nothing is shown after **AllChange Configuration** in the title bar then there are no project has been opened.

Before you can start using **AllChange** you must create a project, see [Creating and Updating Projects](#).

## AC4AC Project

**AllChange** can be used to control the versions of the **AllChange** configuration files that define your Change and Configuration management processes for your **AllChange** Live and Play projects. This is known as AC4AC.

This should be a very simple project with the minimum of configuration and should only be accessible to **AllChange** administrators. You may wish to define a class for the parts which will be controlled to have the **Keep checked out** attribute to ensure when a configuration file is checked in it is checked out again read only ready for use by the **AllChange** project which uses it.

You will also want to define workspaces for each project that the configuration files are to be used for, e.g. your Live and Play projects. In this case define a Live workspace to refer to the Live project directory and a Play workspace to refer to the Play project directory.

You should then keep all configuration files which are to be common between the Play and Live projects under version control in your AC4AC project. You can check out for edit and make modifications and test in the Play workspace (for the Play project). When wishing to promote changes from Play to Live, check in all changed configuration files (remember to check for any new files which are not currently under AC4AC control) and baseline in AC4AC, then use **Update Workspace** to update the Live project/workspace with the changes against the baseline just created.

The AC4AC project should thus have the configuration files under its control checked out read only at all times to the Live and Play (and any other projects required, e.g. Training) unless they are being modified in which case they should be checked out for edit to the Play workspace prior to any modifications using ACCONFIG.

Certain configuration files should be different between your Play and Live projects and these should not normally be maintained under AC4AC control. These files include:

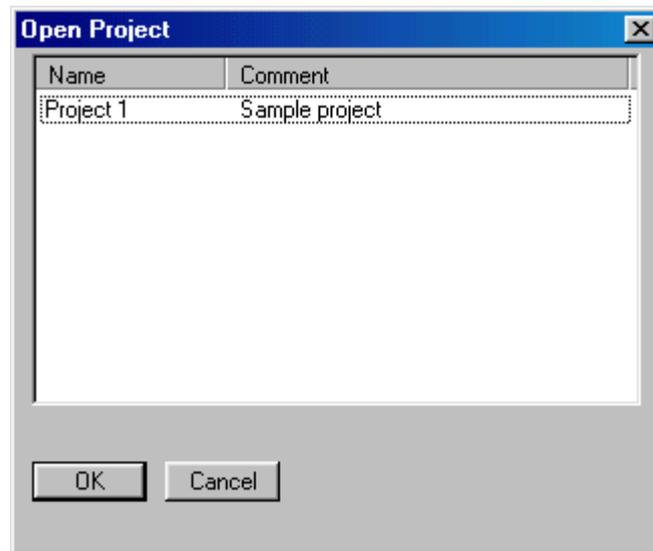
- workspcs.ac, ftpwspc.ac - workspace definitions should be different for different **AllChange** projects referring to different directories
- pools.ac - pool definitions should refer to different directories for different **AllChange** projects
- groups.ac - project specific group definitions may need to be different in Play from Live, for example, to enable testing of groups one would not normally be a member of
- roles.ac - role definitions may need to be different for Play and Live projects, for example, in order to enable testing with roles one would not normally have
- branches.ac - branch definitions may be different for test and Live purposes
- partperm.ac - part permissions are specific to each projects part data

## Opening a Project

The project currently being configured in ACCONFIG is shown in the title bar. The directory containing that project's configuration files is shown in the statusbar at the bottom on the ACCONFIG window, this is the *project directory*.

When ACCONFIG starts it will automatically open a project on startup according what has been selected/specified on the command line. If no projects exist the only options available to you will be to create a new project and open it.

The project to be configured may be changed by using **File | Open Project**.



Having opened a project you can now modify the project configuration for the **Access** control rules, the configuration management **Plan**, the **Workspaces** etc.

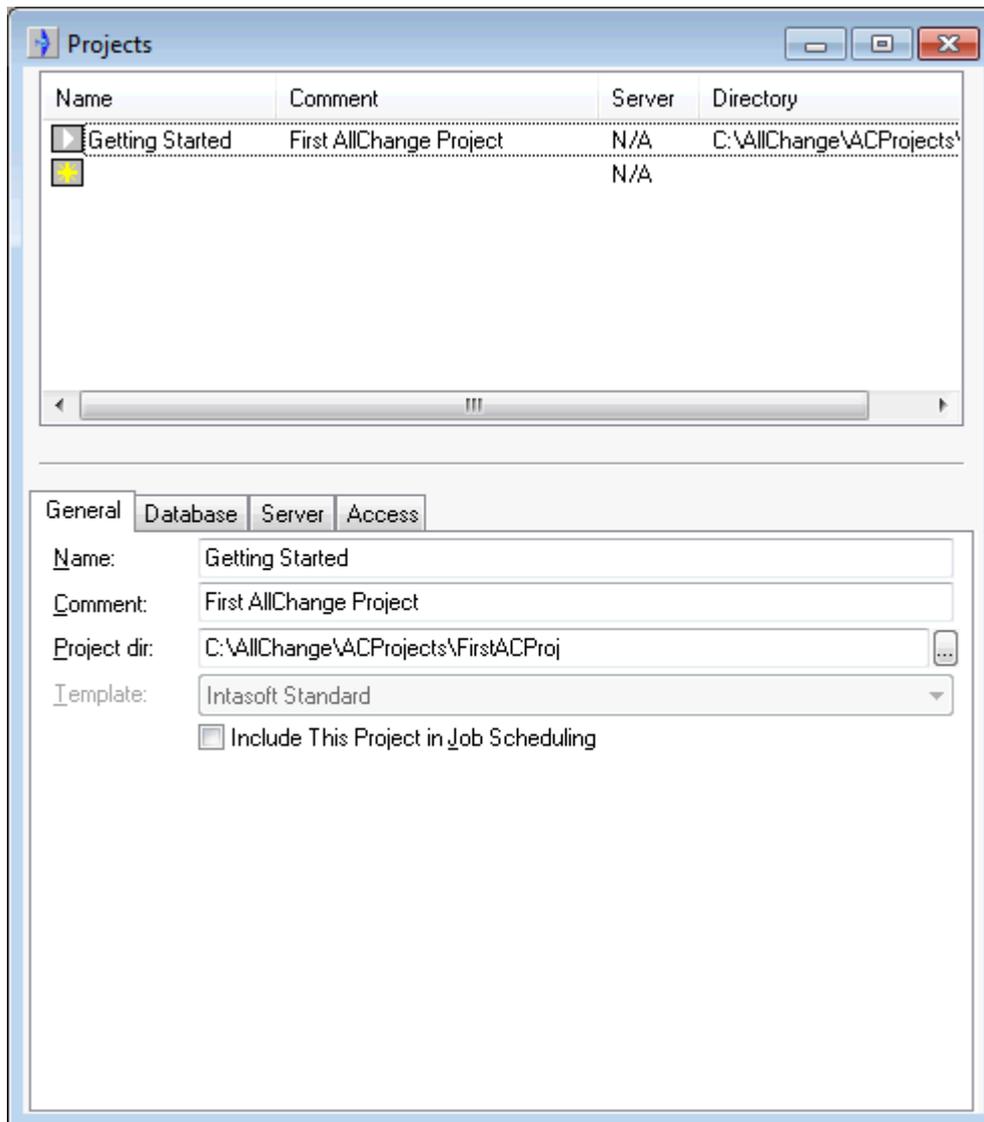
## Creating and Updating Projects

### About Creating and Updating Projects

Project definitions may be created and updated in ACCONFIG using **File | Available projects** in ACCONFIG or selecting **(New Project)** from the project selector on startup.

The first time ACCONFIG is invoked the projects window will automatically open to allow the creation of the first project.

The Project window provides access to the definitions of all of the projects which exist for that **AllChange** version and allows new projects to be created.



for each project the information required is shown on several tabs:

#### General

Provides general information such as the project name and also specifies the project directory

#### Database

Provides information about the SQL database to be used

#### Server

Provides information needed for client/server access to the **AllChange** project directory

#### Access

Specifies what users have access to this project

#### *General*

The General tab provides general information about the **AllChange** project

The screenshot shows a dialog box with four tabs: 'General', 'Database', 'Server', and 'Access'. The 'General' tab is active. It contains the following fields and controls:

- Name:** A text box containing 'Getting Started'.
- Comment:** A text box containing 'First AllChange project'.
- Project dir:** A text box containing 'C:\AllChange\ACProjects\FirstACProj' with a browse button (three dots) to its right.
- Template:** A dropdown menu showing 'Intasoft Standard'.
- Include This Project in Job Scheduling:** An unchecked checkbox.

**Name**

This should be the name by which you wish to refer to the project. This must not include the characters \\* or ? and should start with an alphanumeric character. If the project name is **Default** then this will be the default selection when a list of projects is presented.

**Comment**

This may be used to provide a longer description of the project and will be presented to users on startup of applications when a project needs to be selected

**Project dir**

This must specify the directory in which you wish the **AllChange** configuration files and non data-base data (log files, attachments, version control files) for this **AllChange** project to be stored, this should normally be a directory which does not yet exist. If the directory does not exist ACCONFIG will create it for you. If it does exist then the directory should be a valid **AllChange** project directory . This may be used to define a project where there is already data and configuration files.

If the project directory is to be accessed client/server then this must specify the path to the project directory on the server.

**Template**

This determines the starting *out-of-the-box* configuration that your project will have. This includes class, roles, life-cycles and arbitrary fields definitions, these may then be modified as required for your site. Currently 3 templates are supported out of the box, but site specific templates may be added (see [Project Templates](#) for details):

1. [Intasoft Standard](#): this includes support for source, documents and hardware/assets, different types of change management and an automated and integrated release process
2. [ITIL](#): this supports the IT Infrastructure Library framework for service management, specifically the areas of Problem Management, Issue Management, Change Management, Configuration Management and Release Management in an integrated and automated set of processes
3. [IntaChange Integrate](#): this is a cut down version of the ITIL template specifically designed for integration with IntaChange. It matches in with the equivalent project template in IntaChange.

Once a project definition has been created, the template specified may not be changed, as any site modified project files will be initially based on the template's version of the file. Also, for the same

reason, if more than one project definition is configured for the same project directory, the same template should be specified for each.

### Include this Project in Job Scheduling

This determines whether this project should allow scheduled jobs. Note that for scheduled jobs to be automatically run it is necessary to set up a Windows Scheduled Task - see [Scheduled Jobs](#)

### Database

The database tab provides information about the SQL database to be used

### Use settings for AllChange-installed SQL Express

If checked the system uses the settings which are applied during the installation of the SQL Express instance as part of the **AllChange** setup. These are:

- **SQL Server name:** *computer-name*\AllChange.
- **SQL Server Authentication:** User name: **sa**, Password: **Allchange1**.

i.e. a SQL Server *instance* named **AllChange** on the local machine (*note that this therefore means that **acconfig** must be running on the same machine on which AllChange and SQL Express were installed*), using SQL Server Authentication with a username of **sa** (special administrative user). If for whatever reason you wish to change this (e.g. you decide to change the **sa** user's password), you may do so but you should not then check the **Use settings for AllChange-installed SQL Express** box as that will no longer have the correct information.

### SQL Server name

Name of the SQL Server to use. Instances have a backslash followed by the instance name.

Dropping down the combo box, or clicking the **Refresh** button, will list the available SQL Servers/instances. This requires that the *SQL Server Browser* Windows Service be running on remote machines. Note that this may take a while, especially on slow connections; it may even fail to gather some/all of the remote servers (though it will always show the local ones). It is also possible that some entries do not actually have SQL Servers.

### Authentication

Choose between [Windows](#) or [SQL Server Authentication](#). For SQL Server Authentication, **User**

**name** and **Password** must be filled in.

See also [SQL Server Permissions](#).

#### Database Name

Name of the database for the project. By convention, **AllChange** databases are named **AC\_** followed by the project name (with any spaces or punctuation characters replaced by underscores). This may be changed by typing in another name, but it is recommended that the default is used. Dropping down the combo box, or clicking the **Refresh** button, populates the list with the names of existing (**AllChange**) databases in the SQL Server specified.

#### Encrypt connection

Check the box for data passed between server and clients to be encrypted.

#### Test Connection...

Click to test the result of the settings entered above. If this is a new project it will use the **SQL Server Name** and the **Authentication** to verify that it can connect in order to create a database, but ignore any **Database Name** (since that has not been created yet). If this is an existing project it will also use the **Database Name** to check the connection to the project's database.

If all is well a message will say so. If there is an error the message will give details: inspect this to determine whether the problem lies in reaching the SQL Server, the Authentication credentials, the database or the encryption.

See also [Troubleshooting SQL Server Connection Issues](#).

#### Backup/Restore...

Allows the database to be backed up, or restored from a previous backup. Typically only used for small installations with SQL Express (once in a production environment, it is likely that your SQL Administrators will arrange to perform this). See [Backing Up and Restoring the Database](#).

#### Do not access/update database

Check this box to cause ACCONFIG *not* to access the database in any way when performing actions.

When using ACCONFIG and making changes to projects, in the normal case ACCONFIG will perform necessary actions on the database corresponding to what is being done to the **AllChange** project (and saved in **projects.ac**). For example:

- If you create a new project, it will create the associated database.
- If you delete a project, it will delete the associated database.
- If you rename a project, it will rename the associated database.

This is obviously important to keep the state of the SQL Server "in sync" with whatever is happening to the project definition.

Occasions may arise where you do *not* want ACCONFIG to access/update the SQL Server database. For example:

- You wish to delete an **AllChange** project, but you do *not* want the associated database to be deleted for some reason.
- You have been given an **AllChange** SQL database (from some source), and you have added it into SQL Server. You wish to create a new project to access it, but you do *not* want it to attempt to create the database since it is already present.

In such cases you may check **Do not access/update database** on the [Database](#) tab in ACCONFIG to instruct it not to perform any database operations.

Use care if employing this: it is easy to get ACCONFIG "out of sync" with the database state.

## Server

The Server tab provides information needed for client/server access to the **AllChange** project directory, See [Client/Server](#).

### Server name

This should only be specified if you wish the project directory to be accessed in client/server mode.

This should specify the name of the server that you wish the **AllChange** server to run on and which is to hold the **AllChange** non database data files and configuration files for the project.

### Port number

This should only be specified if you wish this project to be accessed in client/server mode.

This should be blank unless the default port number is not to be used (see [Port Numbers](#)).

### Server proj dir

This should only be specified if you wish this project to be accessed in client/server mode.

This should specify the project directory from the *server's* point of view, see [Setting up client/server — Summary](#) for details of running **AllChange** in client/server mode.

### Compression

Specifies whether the project will compress the data sent over the client/server communication. Compression reduces the amount of data transmitted, and thereby improves network performance. This can be especially noticeable when running over a WAN, but also offers improvements on a LAN. Compression is applied to all data passed between the server and the client: reading of **AllChange** configuration files; up-/down-loading of files during check-in/-out, if running VC client/server.

We suggest sites experiment with this setting on and off to see what difference, if any, it makes. It is also possible to define two projects for the same database, one with compression and one without, to get, say, WAN users with compression and LAN users without. Be aware that compression achieves little or nothing on data which is already compressed, so, for example, transferring ".zip" files will not benefit. If overall you feel that compression does not make much difference in your setup, switch it off.

## Access

The Access tab specifies what users have access to this project

### Access From

#### Any

The project may be accessed from the browser interface or the Windows interface

#### Windows

The project may only be accessed from the Windows interface

#### Web

The project may only be accessed via the Web Browser interface

### User Access

#### Everyone

Specifies that all licensed **AllChange** users may use this project

#### Include named list

Specifies that only those specified in the list of users may use this project. Only these users will be given this project as an option on the project selection list on startup of ACE and ACCONFIG.

Use the **Add/Delete** buttons to select the users who are to have access.

#### Exclude named list

Specifies all licensed **AllChange** users *except* those specified in the list of users may use this project. Only those users permitted to use the project will be given this project as an option on the project selection list on startup of ACE and ACCONFIG.

Use the **Add/Delete** buttons to select the users who are not to have access.

#### No-one

Specifies that no-one may use this project. i.e. it is currently disabled.

To create a new project from the **Projects** window, simply create a new entry and on accepting your changes (i.e. select **Edit | Accept Changes** or **File | Save Window or Project**) you will be offered to create the project directory if it does not exist and will copy the configuration files required into the project directory and create a new database, providing **Do not access/update database** is not checked, otherwise it will expect the database to already exist.

If the project was created as a copy of an existing project (select the existing project, select **Edit | Copy** and then **Paste** it into a new entry), then the configuration files will be copied from the originating project, otherwise they will be copied from the system directory. *Note that this is only the case if a project directory was specified, if none was specified then you will need to copy the configuration files and create the database manually.* An empty database will also be created. This provides a mechanism to create a new project with the same configuration as an existing project but with a new (empty) database.

On successful creation of a new project (whether by copying or creating from scratch) ACCONFIG will ask if you wish to create shortcut icons (on the desktop) for the project.

If you delete a project this will offer to delete the project directory for you and to delete the SQL database unless **Do not access/update database** is checked.

Renaming a project will cause the associated SQL database to also be renamed unless **Do not access/update database** is checked.

If you have created or modified a project to make the **AllChange** project directory accessed client/server then you will need to complete the process by performing the appropriate server installation steps as described in [Setting up client/server — Summary](#)

The project definitions are stored in `projects.ac` in the system directory .

### Initialising the Database

When you create a new project a new database is created in your SQL Server. This database includes information as to what physical files are under **AllChange's** control, what versions of these there are, who created them etc. The actual physical contents of these files and their version history is maintained in external files by the underlying version control tools. The **AllChange** database needs to know where these files are and what they are called. This is determined by the **Location** field of parts in the parts database.

The recommended approach is to determine where you would like the version history files to reside and set the **Location** of the root part (/) to this directory. Allow all further locations to be defaulted and all version history files will be stored within the initially chosen directory hierarchy. (See [Parts, Files and the Database](#) for a more detailed explanation of the relationship between parts and files.)

When a project is first created a subdirectory named `VCFILES` is created in the **AllChange** project directory and the **Location** of the root part is set to point to this directory. If this is not suitable the location may be changed using **ACE and Part | Alter | Alter Location** (specifying / as the **Part**), enter the **Location** required and select **Ok**.

The default location is `$$ACPROJVCDIR\VCFILES`. This uses the "[variable location](#)" feature to mean that the actual location path starts with the value of the `ACPROJVCDIR` environment variable. **ACE** (and **ACC**) automatically create this environment variable and set it to the path to the **AllChange** project directory that they are currently using to access VC files, taking into account whether running VC client/server. This means that the location field should not need to be modified when changing between different client/server access modes or indeed when moving/copying the project directory into different directories.

### About Using AllChange Projects

When any **AllChange** application starts it uses the following rules to determine which **AllChange** project to use:

- If project definitions exist in the system directory
  1. if the command line specifies an explicit `-EACPROJECT=...` the named project is used.
  2. if the command line specifies an explicit `-EACPROJDIR=...`, *or* `-EACSERVER=...` *and* `-EACSERVERPROJDIR=...`, these values override any specified in the project definition.
  3. if only one project is defined it is used automatically
  4. if only one project is available to the user based on his username, it is used automatically

5. if entering a GUI program (ACE, ACCONFIG) the user is prompted to select from the available projects, with his default project initially selected;

The Administrator can set (via ACCONFIG) which projects will be offered to a given user

- If no project definitions exist in the system directory, an error will be raised. The only exception to this is that ACCONFIG may start without this information in order to create the first project.

The information about project definitions is held in the **AllChange** system directory. The system directory is determined either from an explicit `-EACDIR=...` command line argument or from the workstation's registry. The registry entry is set up for you by the workstation installation. You can therefore use the command line argument to override the default system directory.

Note that if running with project directory client/server access this is the only time that a client *will* require direct access to the system directory

## Project Templates

### Project Templates

**AllChange** supports the concept of project templates. These are predefined sets of AllChange configuration files which may be used as the starting point for creating a new project.

Intasoft currently supplies 3 project templates:

1. [Intasoft Standard](#): this includes support for source, documents and hardware/assets, different types of change management and an automated and integrated release process
2. [ITIL](#): this supports the IT Infrastructure Library framework for service management, specifically the areas of Problem Management, Issue Management, Change Management, Configuration Management and Release Management in an integrated and automated set of processes
3. [IntaChange Integrate](#): this is a cut down version of the ITIL template specifically designed for integration with IntaChange. It matches in with the equivalent project template in IntaChange.

Additional project templates may be created according to site specific requirements. Simply place the configuration files to be used for any project created using the template in a subdirectory of the `projtmpl` subdirectory of the system directory and add the template information to the [projtmpl.ac](#) file.

For further information on example life-cycles in the project templates see:

[Standard Template Cycles](#)

[ITIL Template Cycles](#)

[IntaChange Integrate Cycles](#)

### Intasoft Standard Configuration

#### *About Standard Configuration*

**AllChange** comes with ready to use *out-of-the-box* configurations which can be used as they stand or can be tailored to individual requirements.

The Standard out-of-the-box configuration has been created with the experience of many different types and sizes of users of **AllChange** and even if you need to tailor it, it will provide a very good starting point.

The configuration comprises a set of fully integrated processes from Change Management to Release covering:

#### Parts

- [Part Classes](#)
- [Part Fields](#)
- [Part Change Process](#)

#### CRs

- [CR Classes](#)
- [CR Fields](#)
- [CR Change Management Process](#)

### Baselines

- [Baseline Classes](#)
- [Baseline Fields](#)
- [Baseline Release Management Processes](#)

### [Integrated Process Overview](#)

## Standard Parts

### Part Classes for Standard Project Template

The standard configuration defines the following classes of part:

#### Source

Defined for components which are to be modified by checking in and out the part directly (i.e. not via a life-cycle). This may be used for source code, scripts, other deliverables etc. Components of this class are not subject to formal change control i.e. they may be created and modified without identifying a CR to authorise the change and associate with the change automatically. However they must have a **Comment** specified on check in — see [Part Fields](#)

#### ControlledSource

Defined for components which are to be modified by checking in and out using the **source\_cycle** associated with them but are also subject to formal change control, i.e. they may not be created or modified without identifying a software CR (SCR) to authorise the change and associate with the change automatically.

#### Document

Defined for components which are not source code which are to be modified by checking in and out the part directly (i.e. not via a life-cycle). Components of this class are not subject to formal change control i.e. they be created and modified without identifying a document CR (DCR) to authorise the change and associate with the change automatically. However they must have a Comment specified on check in - see [Part Fields](#)

#### ControlledDoc

Defined for components which are to be modified by checking in and out directly (i.e. not via a life-cycle) but are also subject to formal change control, i.e. they may not be created or modified without identifying a CR to authorise the change and associate with the change automatically.

#### ReviewableDoc

Defined for components which are not subject to formal change control but require a review process for each component to approve the changes. Components of this class have the **review\_cycle** associated with them but are not subject to formal change control i.e. they may be created or modified without identifying a CR to authorise the change and associate with the change automatically.

#### SolidWorksDoc

Defined for components which are SolidWorks documents (parts and assemblies) and used by the [integration with SolidWorks](#). Components of this class are subject to formal change control and require a review process for each component to approve the changes. Components of this class have the **SWDoc\_cycle** associated with them.

#### SolidWorksDrawing

Defined for components which are SolidWorks drawings and used by the [integration with SolidWorks](#). Components of this class are subject to formal change control and require a review

process for each component to approve the changes. Components of this class have the **SWDoc\_cycle** associated with them.

#### **Derived**

Defined for components which are derived or which are supplied by a third party and for which the source is not available.

Derived components have the `derived_cycle` associated with them which is used to import new supplied versions and are also subject to formal change control, i.e. they may not be added nor a new version submitted without a CR authorising the change.

#### **Websrc**

Defined for components which are to be modified by checking in and out via the `web_cycle` associated with them. This may be used for web code, HTML pages etc. Components of this class are not subject to formal change control.

#### **ITAsset**

Defined for component which are IT Assets to be recorded as an asset register with no file associated with them. Components of this class have the `it_asset_cycle` and are the subject of formal change control i.e they may not be created or a new version created without identifying an IT Asset CR (ITCR) to authorise the change and associate with the change automatically.

#### **HWComponent**

Defined for components which are hardware/manufacturing components with no file associated with them. Components of this class have no life cycle and are the subject of formal change control i.e they may not be created or a new version created without identifying an engineering CR (ECR) to authorise the change and associate with the change automatically.

#### **Doors**

Defined for components which are DOORS modules and used by the [integration with DOORS](#). Components of this class have no life-cycle and are not subject to formal change control.

### **Part Fields for Standard Project Template**

The following part arbitrary fields are defined in the standard configuration:

#### **Deliverable**

This is defined for `ControlledSource` class parts. It is used to indicate whether the part is a part of the deliverable product. It is used by the supplied release process to determine what components to release. This should be used for objects which form part of a product/ release and are delivered in the form in which they are stored in **AllChange** - i.e they do not undergo a transformation process and the derived object is what is released (e.g source  $\rightarrow$  compiled object)

#### **Description**

This is defined for all classes of part. It is used to hold a description of the part.

#### **Reviewer**

This is defined for components of class **ReviewableDoc**, **SolidWorksDoc** and **SolidWorksDrawing**. It should be used to specify the name of the user who it to be the document reviewer. This is compulsory and will be prompted for on creation of the part.

#### **No Doc Stamp**

This is defined for components of class **ControlledDoc**, **Document** and **ReviewableDoc**. If ticked the document will not be checked for any AllChange fields to be stamped. This should be used for all components which are Word or Excel documents where it is known that document stamping is not used in order to improve performance.

**Manufacturer**

This is defined for parts of class **ITAsset** and may be used to hold who the manufacturer of the IT asset is.

**Supplier**

This is defined for components of class **HWComponent** and may be used to hold the who the supplier of the item is.

**Comment**

This is defined for versions of components for all classes which do not require a CR in order to be checked out for edit. This is compulsory and will be prompted for on check in.

**Notes**

This is defined for versions of components of class **ITAsset** and **HWComponent**. This is compulsory and will be prompted for on creating a new version.

**Location**

This is defined for versions of components of class **ITAsset**. This may be used to store the physical location of the asset.

**Cost**

This is defined for versions of components of class **ITAsset** and **HWComponent**. This can be used to record the cost of the item.

**ArchType**

This is defined for versions for all classes which have a file and indicates the type of archiving. This may be OnLine, OffLine or empty.

**Merge From**

This is defined for versions for components of any class and is used to hold information as to which versions this version was merged from. This is set automatically when using the AllChange Merge facility

**Issue No**

This is defined for versions for components of **ReviewableDoc** class and is auto incremented as the version progresses through the life cycle. The **Revision Number attribute** is used to specify a revision number of the form *Major.Minor* where *Major* is the Major **Revision Counter** and *Minor* is the MinorNumber Revision Counter. This implements a revision numbering scheme of 1.0, 1.1.... whilst the document is in Draft, once it is approved it becomes 2.0, the next draft would then be 2.1 etc. This allows a document revision number to be stored separately from the **AllChange** version number.

**Revision**

This is defined for versions for components of **SolidWorksDoc** and **SolidWorksDrawing** classes and is auto incremented as the version progresses through the life cycle. It is synchronised with the Revision custom property in SolidWorks. The **Revision Number attribute** is used to specify a revision number of the form *MajorMinor* where *Major* is the Major **Revision Counter** and *Minor* is the MinorAlpha Revision Counter. This implements a revision numbering scheme of 1A, 1B.... whilst the document is in Draft, once it is approved it becomes 1, the next draft would then be 2A etc. This allows a document revision number to be stored separately from the **AllChange** version number.

**Part No**

This is defined for versions for components of **SolidWorksDoc** and **SolidWorksDrawing** classes. It is synchronised with the Part No custom property in SolidWorks.

In addition the following fields are defined for instances of versions of parts

**Notes**

This is defined for instances of versions of parts of class **HWComponent**.

### Serial Number

This is defined for instances of versions of parts of class **HWComponent**. This may be used to store the serial number for an instance.

### Customer

This is defined for instances of versions of parts of class **HWComponent**. This may be used to hold the customer to whom the instance has been supplied

## Part Change Processes

### About Part Change Processes for Standard Project Template

For all components they may be modified using one of two methods:

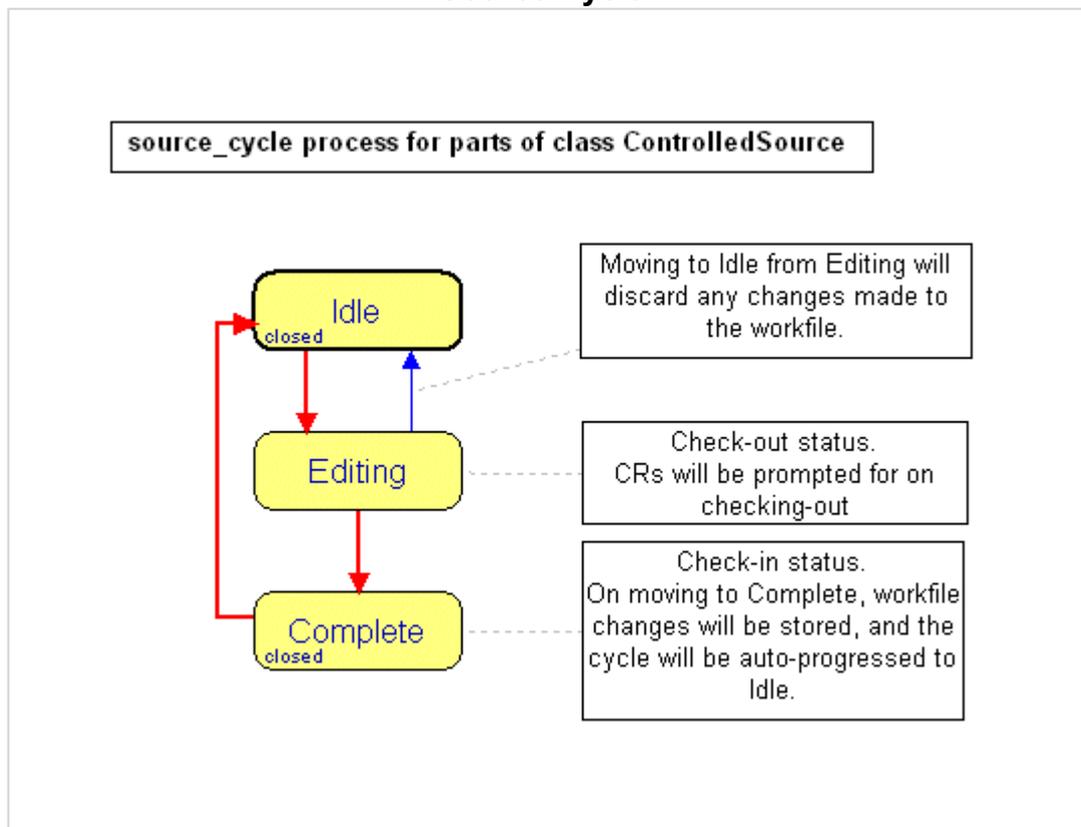
- **Check Out/Check In** uses issue/return command directly
- **Check Out/Check In** makes use of the life-cycle to do the issue/return commands for you

Which of these methods is used is determined by whether there is a life-cycle associated with a particular part class .

### ControlledSource Cycle Driven Change Process

The **ControlledSource** class of component is associated with the **source\_cycle** life-cycle, this is shown in the [Source Cycle](#) diagram below.

### Source Cycle



In order to make a change the required version should be **checked out**. The **Check Out** command will change the status of the version from *Idle* to *Editing*, this will create a new version and check out the version for edit into the current workspace . This may be achieved by using **Check Out** or double clicking on

status in the Cycle View window. *Check Out* statuses should **not** be set by changing the status explicitly using the *Status* button on the *Part Viewer*.

If the component is of class **ControlledSource** then the CRs which authorise the change must be specified. The classes of component which are subject to formal change control in this way is specified as an attribute of the class.

The CRs will be prompted for by **Check Out**.

Only those CRs which are valid for authorising a change will be shown. This will include CRs which are assigned to you and are of class either **SCR** in a status of *InWork* or **DCR, ECR or ITCR** in a status of *UnderWork*. The class of the CRs and the status that they should have to authorise the change is defined in the CR life-cycle definition.

The CR numbers authorising the change will be stored in the check-out record's comment field. Also, the parts checked out will be added to the CRs' partsaffected. This means that it is possible to see which parts are, or have been, checked out against a CR by looking at the partsaffected, unless parts have been manually removed from the CR's partaffected.

When you have finished making your changes and wish to check your code/document back in, either use **Check In** which will progress the status to *Complete*, or double click on the *Complete* status in the Cycle View Window. This will return the component from edit, storing the new version under control.

If the component was checked out against one or more CRs, it will also associate the new version with the CRs against which it was edited, as both partsaffected by each CR if it is not already one, and the new version as the versionsolved by each CR. If a new version was not checked-in, or if the edit was discarded, then the component is removed as a partaffected providing there is not already a version solved for the part.

If a component version is checked out for edit (the status of a component version is *Editing*) and you wish to discard any changes you have made and effectively change your mind about modifying the component, then **Check In** may be used and select the **Discard** option, or the status may be changed from *Editing* back to *Idle*, and this will effectively reverse the status change to *Editing* as though it had never happened. When a version has been *Completed*, its status will be automatically set back to *Idle* ready for further changes.

## ReviewableDoc Cycle Driven Change Process

The **ReviewableDoc** class of component is associated with the [review-cycle](#).

This cycle is intended for parts that need to undergo a review procedure, that is, all changes to the part must be approved by a reviewer. In addition ReviewableDoc parts have an auto incremented issue numbering field called **Issue No**. This implements an issue numbering scheme of 1.1, 1.2..... whilst in draft progressing to 2.0 on approval. This allows a separate document issue number from the AllChange version number.

The Reviewer for the document is specified on an arbitrary field named **Reviewer**.

When a new document is checked in the user is prompted to select whether the first version of the document is a Draft or is Approved .

Draft versions may be checked out for modification (this will enter the Editing status and the Minor revision counter is incremented) or progressed to the Review state indicating that the document version is ready for review. On entry to Review the reviewer will be mailed to inform him/her that the document is ready to be reviewed.

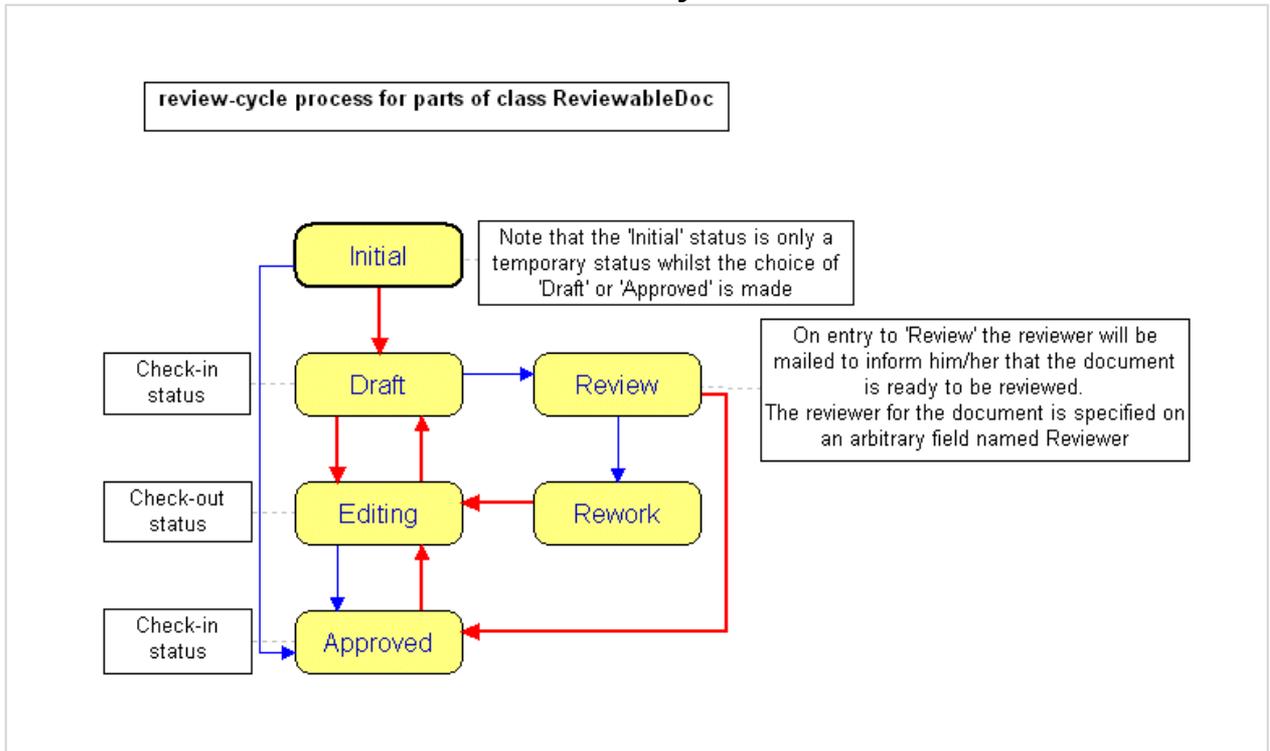
When a checked out document is checked back in a choice may be made as to whether the new version is a Draft or is Approved. Note that only the document reviewer may check it in as Approved.

When a reviewer has reviewed the document it may be either Approved or set to Rework. If Rework is required then the author will be emailed to inform them that further work is required. The document may then be checked out for editing and modifications made.

When the document is approved the Major revision counter is incremented and the Minor counter is reset to 0.

In the diagram below the life-cycle is shown. Note that the Initial status is only a temporary status whilst the choice of Draft or Approved is made.

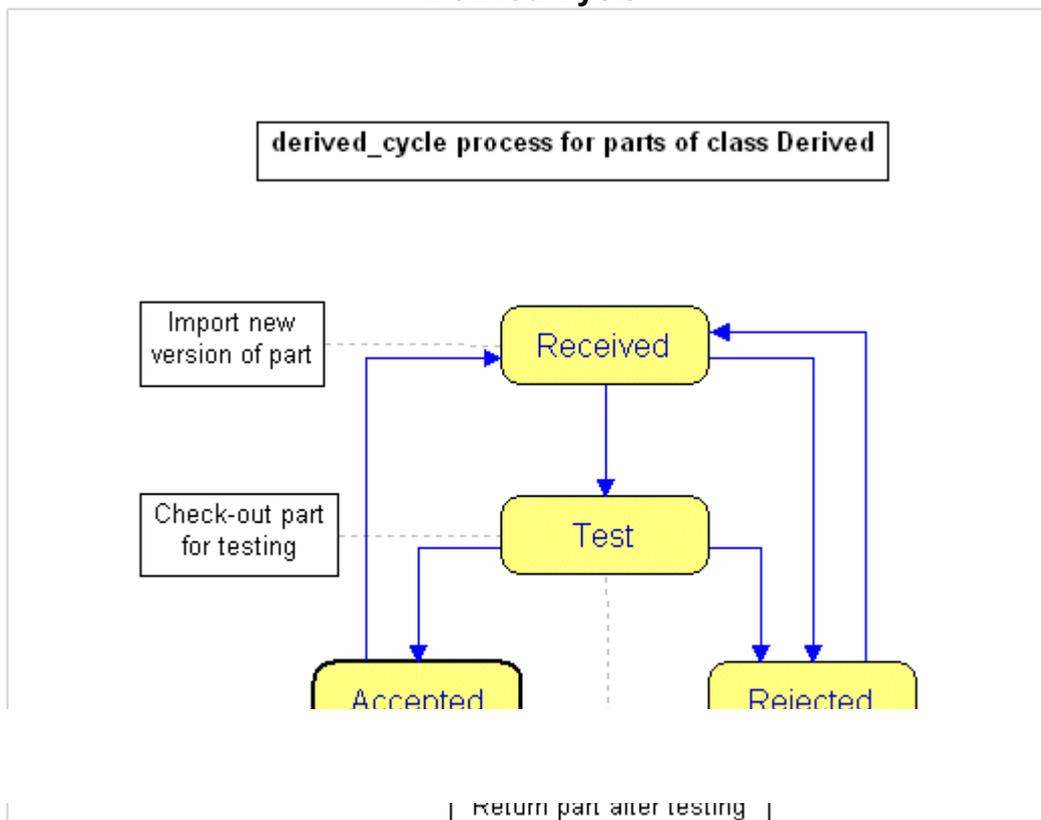
### Review Cycle



### Derived Cycle Driven Change Process

The **Derived** class of component is associated with the life-cycle shown in the [Derived Cycle](#) diagram below:.

### Derived Cycle



Initially an *Accepted* version of a component is added to the **AllChange** database. When a new version of the component is received the file should be placed in a directory which is an **AllChange** workspace and that workspace attached to.

The status of the component is changed to *Received*. If the component is of class **Derived** or any other class which requires a CR to authorise a change, then the CR authorising the change should be specified (see ControlledSource Cycle Driven Change Process for further details).

On entering the status *Received* the contents of the new version of the file will be stored under **AllChange**'s control and the workfile will be removed from the workspace directory. Also any CRs specified will be associated with the new version of the component.

Once *Received* the component may be immediately rejected by changing the status to *Rejected*. Alternatively the component may be tested by changing the status to *Test*. This will check out the component for read to the current workspace for any testing to take place. When testing is completed the status should be changed to *Accepted* or *Rejected*. In either case the component will be checked in from the workspace. The component is now ready for the next version to be *Received*.

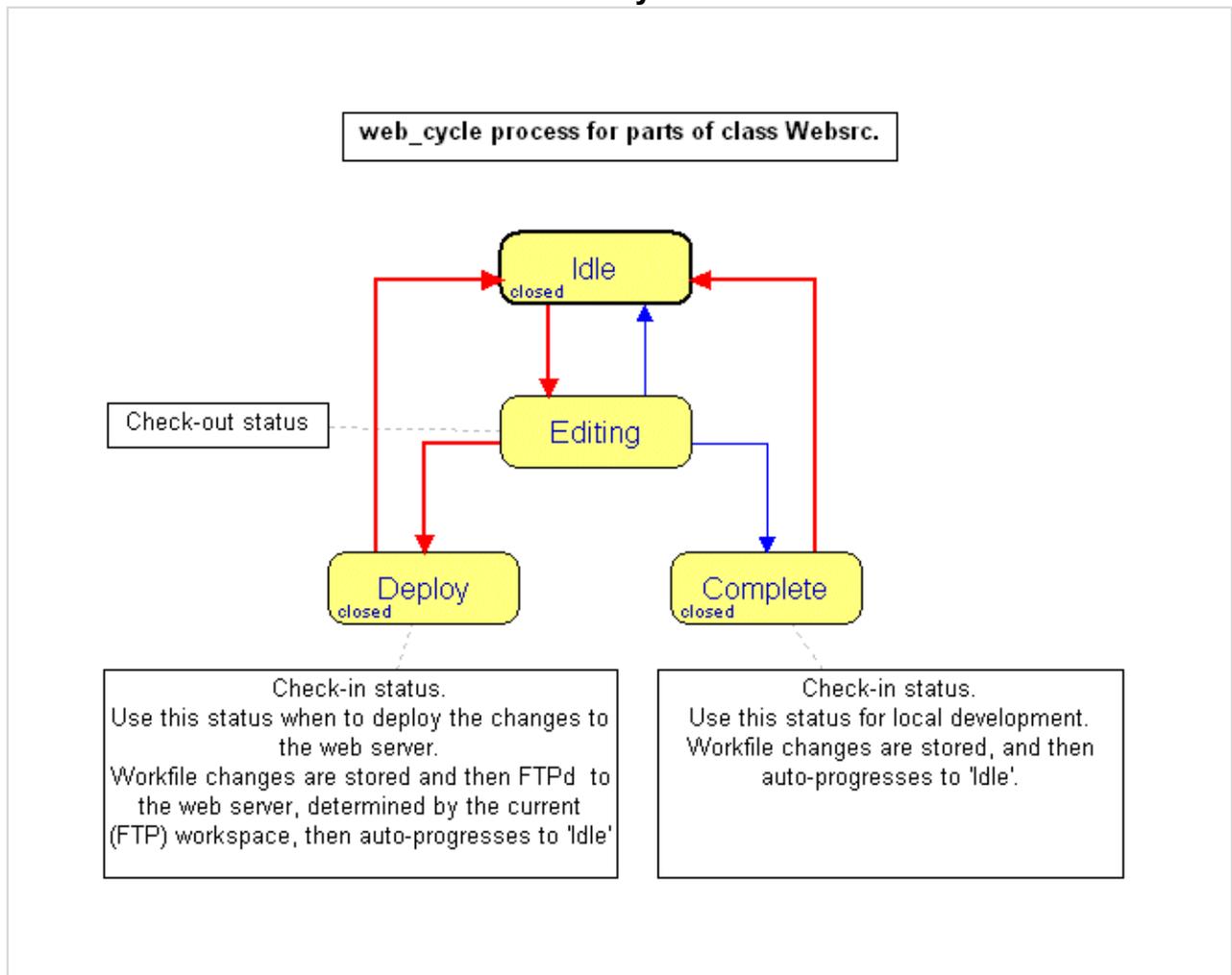
Note that the **Check In** command will not make use of the life-cycle for **Derived** parts as the component has not been checked out. In order to make use of the **derived** cycle, the status changes must be made explicitly.

This change process is very useful for components which are supplied by a third party and are not as such edited.

## Websrc Cycle Driven Change Process

The **Websrc** class of component is associated with the [web\\_cycle](#) life-cycle shown below.

### Web Cycle

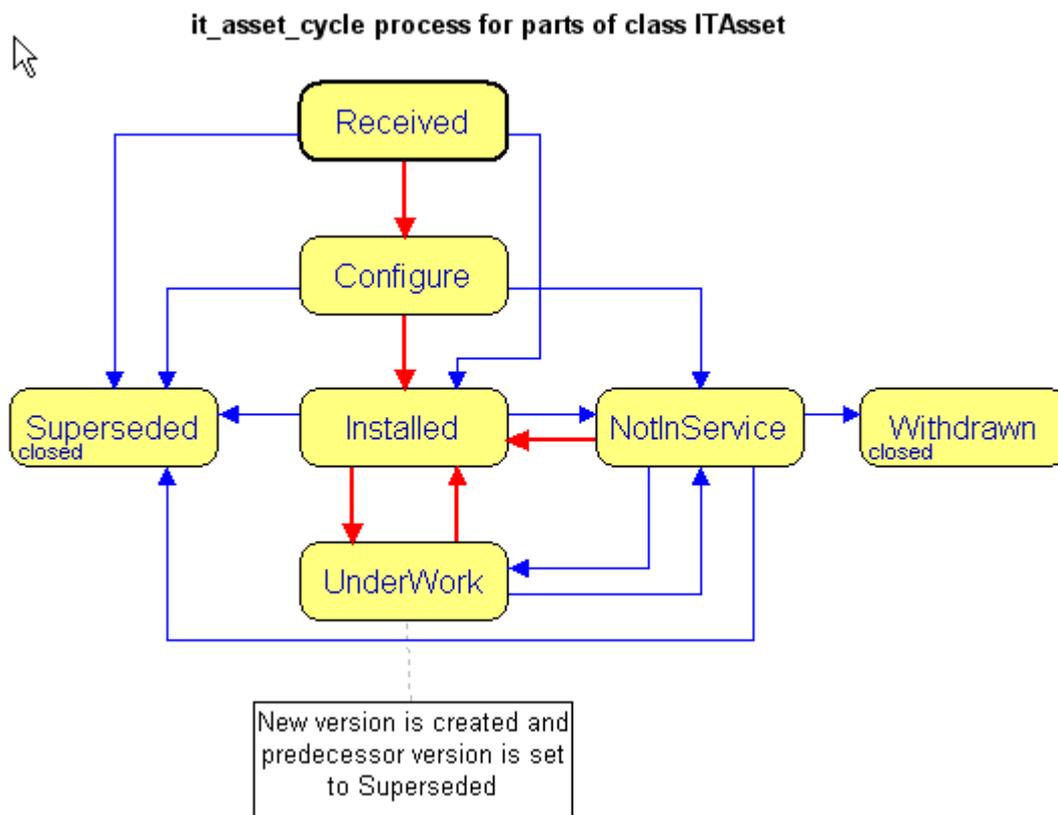


This process is intended for use when developing HTML pages for use in an Intranet. The basic change process is identical to that for **ControlledSource** items. The only difference is the *Deploy* state which is used when you wish to deploy your changes to the web server. Until then you should develop your HTML code locally using the check in/out process and selecting the *Complete* status on check in. In order for the FTP deployment to work the developments must be done when attached to a workspace which is defined as an FTP workspace for type *deploy* and defines the web server which is to be used. When you are ready to deploy your changes to the web server, change the status to *Deploy* (**Check In** and select *Deploy*) and the component will be checked in, then it will be extracted and will have its links changed to reflect the new webserver location according to those defined in the [Web Map](#) and then will be FTP'd to the required web server location.

### ITAsset Cycle Driven Change Process

The **ITAsset** class of component is associated with the [it\\_asset\\_cycle](#) life-cycle shown below.

#### IT Asset Cycle



This process is for use when using **AllChange** to store information about IT Assets as parts with the **No File** flag. It allows the process of an asset being received through to becoming withdrawn or no longer used to be tracked.

Initially the asset is *Received*, it may then require some configuration, in which case it can be progressed to *Configure*.

Having been configured or if no configuration is needed it may then be progressed to *Installed* to indicate it is now up and running and being used.

Once installed the asset may then go through a number of changes (e.g. A PC may have software installed on it, its memory upgraded etc). This may be recorded through the *UnderWork* status which will cause a new version to be created and the predecessor version to have its status set to *Superseded*. In order to enter *UnderWork* a CR must be specified as the authority for the change if the part is of class **ITAsset** or

any other class which requires a CR to authorise a change. When the work is complete it may then be set to *Installed* to denote that the new version is up and running and being used.

At any time when the asset is *Installed* it may be set to *NotInService* to denote that it is no longer in use but is still available. *NotInService* assets may be brought back into service by setting their status to *Installed*, or may be *Withdrawn* entirely

## SolidWorks Document Cycle Driven Change Process

The **SolidWorksDoc** and **SolidWorksDrawing** classes of component are associated with the **SWDoc-cycle**.

This cycle is intended for parts which are SolidWorks documents and that need to undergo a review procedure, that is, all changes to the part must be approved by a reviewer. In addition these parts have an auto incremented revision number field called **Revision**. This implements a revision numbering scheme of 1A, 1B..... whilst in draft progressing to 1 on approval. This allows a separate document revision number from the **AllChange** version number.

The Reviewer for the document is specified on an arbitrary field named **Reviewer**.

When a new document is checked in the user is prompted to select whether the first version of the document is a Draft or is Approved .

Draft versions may be checked out for modification (this will enter the Editing status and the Minor revision counter is incremented) or progressed to the Review state indicating that the document version is ready for review. On entry to Review the reviewer will be mailed to inform him/her that the document is ready to be reviewed.

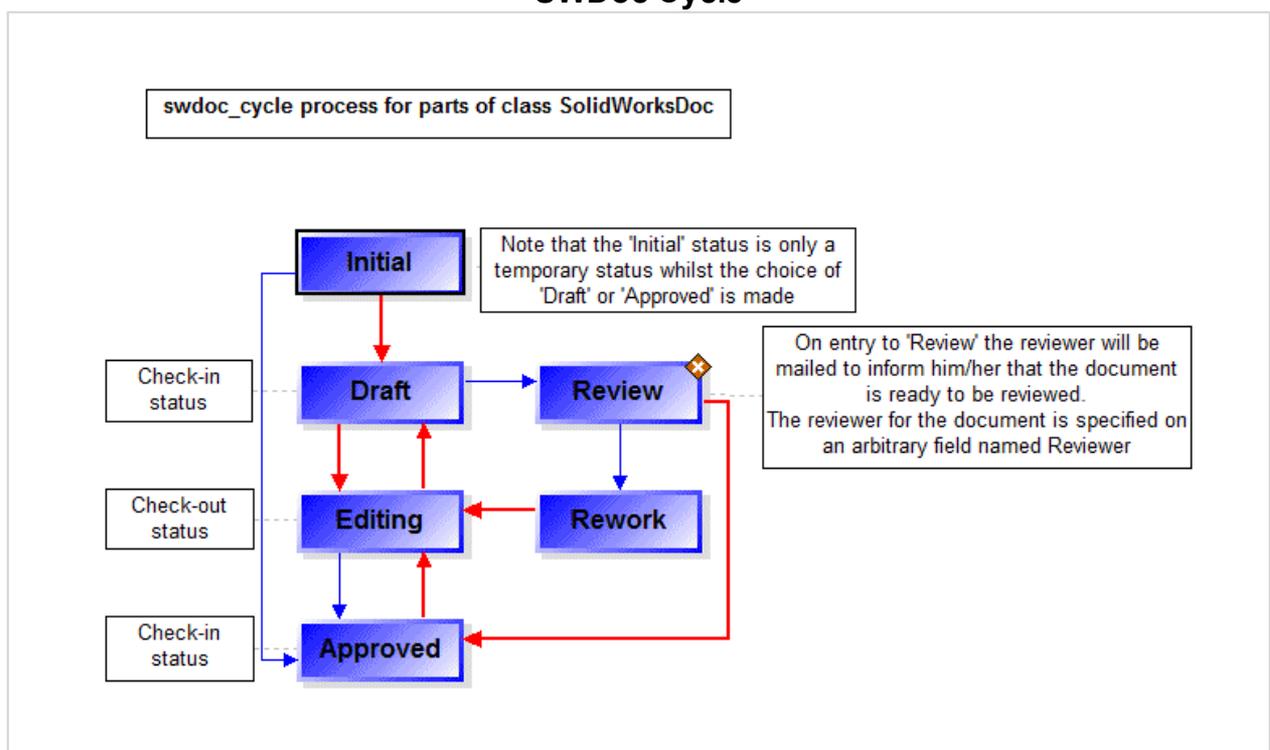
When a checked out document is checked back in a choice may be made as to whether the new version is a Draft or is Approved. Note that only the document reviewer may check it in as Approved.

When a reviewer has reviewed the document it may be either Approved or set to Rework. If Rework is required then the author will be emailed to inform them that further work is required. The document may then be checked out for editing and modifications made.

When the document is approved the Minor revision counter is blanked out.

In the diagram below the life-cycle is shown. Note that the Initial status is only a temporary status whilst the choice of Draft or Approved is made.

### SWDoc Cycle



## Change Process with no Life-Cycle

If you do not wish your change process to be cycle driven then you may use the classes which do not have a life-cycle associated with them and the **Check Out** and **Check In** will use issue and return commands directly instead.

When you wish to check out a part for edit you should use **Check Out** and when you have completed your changes you should **Check In**. Components of class **Source**, **document**, and **ControlledDoc**, for example, will be modified in this way.

Any components of class **ControlledDoc** require that a CR is specified authorising the change. This will be prompted for as described for **ControlledSource**. The classes of component which are subject to formal change control in this way are specified as an attribute of the class. As for changing status to Complete for the life-cycle approach, on returning from edit the CRs against which the check out was done will be associated with the new version of the component.

### Standard CRs

#### CR Classes for Standard Project Template

The standard configuration defines the following classes of Change Requests (CR).

##### DCR

Defined for CRs for changes to documents, to be used with parts of class **ControlledDoc**. DCRs are associated with the **generic\_cr\_cycle** life cycle. CRs of class **DCR** are numbered with a DCR prefix followed by a five digit auto incrementing number, e.g. **DCR00001**.

##### SCR

Defined for CRs for changes to software, to be used with parts of class **ControlledSource**. SCRs are associated with the **scr\_cycle** life-cycle. CRs of class **SCR** are numbered with an SCR prefix followed by a five digit auto incrementing number, e.g. **SCR00001**

##### ECR

Defined for CRs for changes to engineering components, to be used with parts of class **HWComponent**. ECRs are associated with the **generic\_cr\_cycle** life\_cycle. CRs of class **ECR** are numbered with an ECR prefix followed by a five digit auto incrementing number e.g. **ECR00001**

##### ITCR

Defined for CRs for changes to IT Assets, to be used with parts of class **ITAsset**. ITCRs are associated with the **generic\_cr\_cycle** life-cycle. CRs of class **ITCR** are numbered with an ITCR prefix followed by a five digit auto incrementing number e.g. **ITCR00001**

##### WorkOrder

Defined for CRs for manufacturing, to be used with baselines of class **Assembly** and **ProductionItem**. WorkOrders are associated with the **wo\_cycle** life-cycle. CRs of class **WorkOrder** are numbered with a WO prefix followed by a five digit auto incrementing number e.g. **WO00001**

These classes illustrate change management for Software, Documents, IT Assets and Engineering changes as well as manufacturing work orders.

#### CR Fields for Standard Project Template

Various arbitrary fields are defined for the different classes of CR in the standard configuration - these may be modified according to requirements. There are many more arbitrary fields which may be defined as required. The standard definitions include many common fields that people use on their change request documents and are intended to provide some ideas as to the sort of information you may wish to maintain.

The standard fields are:

Field Name	Class	Description
<b>Priority:</b>	Any	denotes the priority that the CR has, values are: <b>High</b> <b>Medium</b> <b>Low</b>
<b>Locked:</b>	Any	indicates whether the CR is locked against further change
<b>Type:</b>	SCR	indicates the type of change that is required, values are: <b>Bug</b> <b>Error</b> <b>Off Spec</b>
<b>Complexity:</b>	SCR, ECR, ITCR	the complexity the change will require, values are: <b>High</b> <b>Medium</b> <b>Low</b>
<b>Keywords:</b>	SCR	this may be used to specify keywords to be used for search purposes
<b>Introduced:</b>	SCR	indicates the stage of development during which the problem was introduced, values are: <b>Analysis</b> <b>Design</b> <b>Build</b> <b>Test</b>
<b>Discovered:</b>	SCR	indicates the stage of development during which the problem was discovered. Values are the same as <b>Introduced</b> . Analysis of this together with the Introduced field may be used to provide some quality metrics
<b>Date Due:</b>	SCR, ECR, ITCR, DCR, WorkOrder	may be used to specify the date by which the work for this CR should be completed
<b>Target Release:</b>	SCR, DCR	may be used to indicate the release for which this change is targeted
<b>Project :</b>	SCR, DCR, ECR	indicates the project associated with the change
<b>Problem Source :</b>	ECR	indicates the source of the problem
<b>Raised By:</b>	ITCR	the person that raised the request
<b>Estimated Effort:</b>	ITCR	estimated number of days to make this change
<b>Actual Effort:</b>	ITCR	actual number of days to make this change
<b>Quantity:</b>	WorkOrder	The number of items that are to be

**Quality Control:** WorkOrder manufactured  
 Whether this WorkOrder is the subject of quality control procedures

In addition the following fields are defined for the CR Items Affected:

**Description:** A description of the relationship  
**Obsolete:** Whether the relationship was created due to the item being made obsolete. This is set to Yes when a controlled part is made obsolete against a CR.

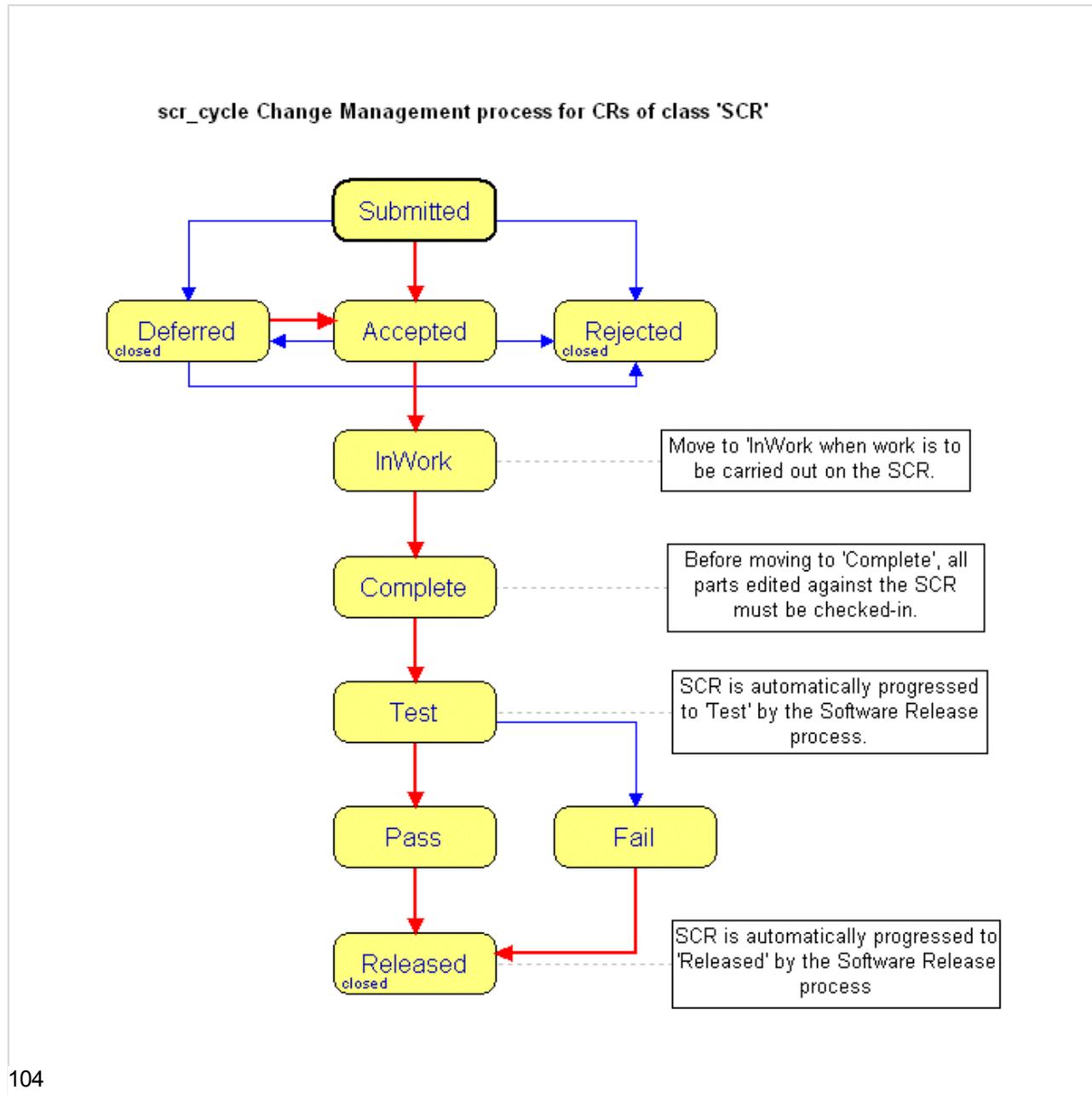
**CR Change Management Process**

**About CR Change Management Process for Standard Project Template**

The change management process is implemented by the CR life-cycles defined for each class of CR. This is used to determine which logged CRs should be accepted for work and then allocated to users for the work to be done. It is also used to indicate what progress has been made.

**SCR Change Management Process**

The scr\_cycle implements a change process for Software CRs. This is used by the SCR (Software Change Request) class of CR.



Initially an **SCR** is *Submitted* reporting some fault or change requirement. Some analysis is then performed to decide whether the SCR should be *Accepted*, *Rejected* or *Deferred*.

Once an SCR has been *Accepted* it may then be assigned to an individual for work.

When the assignee is ready to start work on the SCR the status is changed to *InWork*. This then makes this a valid CR for change when checking files out for edit.

When all changes have been made for the CR the status may be set to *Complete* to indicate that the work has been completed.

Before the *InWork* status may be exited:

- All parts associated with the CR must have been changed (i.e. all parts affected by the CR must have a version solved)
- All versions solved associated with the CR must have been checked in
- There must be no parts checked out for edit against the CR (i.e. there must be not parts checked out for edit with the CR in the check-out comment)

SCRs which are in a status of *Complete* are ready for incorporation into a release (baseline) of class [SWRelease](#).

When the changes have been incorporated into a release and are ready for testing, then the status is progressed to *Test* automatically by the baseline.

The SCR should then be tested and the status progressed to *Pass* or *Fail* accordingly.

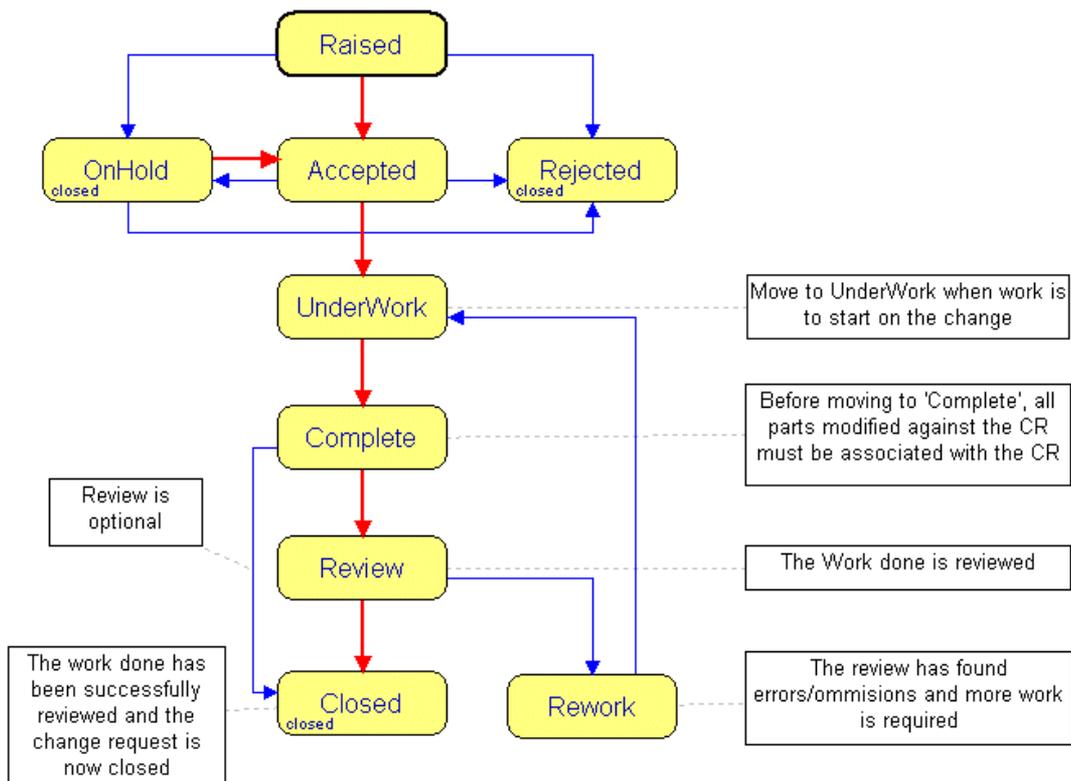
Once the SCR has been tested it may then be released. The status will automatically be progressed to *Released* by the baseline when the release has been made.

It may seem strange to allow failed CRs to be released, however, this has been deliberately allowed as there are times when known problems still have to be released. This should however, only be done when necessary.

## Generic Change Management Process

The `generic_cr_cycle` implements a generic change request process which is used by the DCR (Document Change Request), ECR (Engineering Change Request), ITCR (IT Asset Change Request) and DoorsRequirement classes of CR.

**generic\_cr\_cycle** Change Management process for CRs of class ECR, DCR, ITCR, DoorsRequirement



### Generic CR Cycle

Initially the CR is *Raised* reporting some fault or change requirement. Some analysis is then performed to decide whether the CR should be *Accepted*, *Rejected* or *Deferred*.

Once the CR has been *Accepted* it may then be assigned to an individual for work.

When the assignee is ready to start work on the CR the status is changed to *UnderWork*. This then makes this a valid CR for change when checking files out for edit or creating new versions of parts.

When all changes have been made for the CR the status may be set to *Complete* to indicate that the work has been completed.

Before the *UnderWork* status may be exited:

- All parts associated with the CR must have been changed (i.e. all parts affected by the CR must have a version solved)
- All versions solved associated with the CR must have been checked in
- There must be no parts checked out for edit against the CR (i.e. there must be not parts checked out for edit with the CR in the check-out comment)

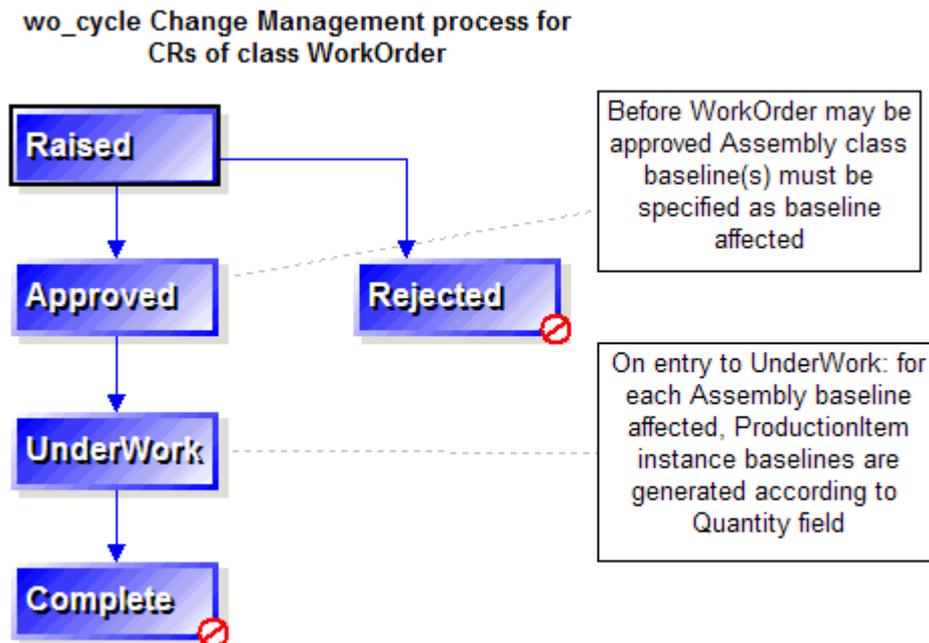
CRs which are in a status of *Complete* are ready for incorporation into a release (baseline) using **CRs For Baseline**.

When the changes are to be reviewed the status is progressed to *Review* and the CR can be assigned to an appropriate reviewer.

Once reviewed the status should be progressed to *Closed* if the changes are acceptable and to *Rework* if further work is needed whence it may be reassigned to the person who did the work and moved back through the cycle.

## Work Order Change Management Process

The `wo_cycle` implements a change process for Work Order CRs for a manufacturing process. This is used by the **WorkOrder** class of CR.



Initially the WorkOrder is *Raised* reporting a requirement to manufacture one or more Assemblies. The Assemblies to be manufactured should be specified as baselines affected. These may be meta or part release baselines which should be of class Assembly. These baselines should identify the part versions which are to be manufactured.

Before the WorkOrder may be approved at least one Assembly baseline affected must be specified.

Once *Approved* the quantity of assemblies to be manufactured should be specified on the **Quantity** field prior to progressing to *UnderWork* to indicate that the items are being manufactured.

On entry to *UnderWork*, for each Assembly baseline affected, **Quantity** ProductionItem class instance baselines will be created containing new instances of the versions specified in the Assembly baselines. The ProductionItem class baselines are associated with the WorkOrder as baselines solved.

The ProductionItem baselines will be given the same basename as the corresponding Assembly baseline and will have a version of:

```
Instance.<number>
```

When the manufacturing process is complete the WorkOrder may be progressed to *Complete*.

### Standard Baselines

#### Baseline Classes for Standard Project Template

The standard configuration defines the following classes of baseline.

##### SWRelease

Defined for baselines of software (usually source code) which are to be processed using the release process implemented by the `sw_release_cycle`.

##### Assembly

Defined for baselines of hardware used in manufacturing to define the parts which make up an

assembly. May be used for part or meta baselines. When used in conjunction with WorkOrder class CRs these may be used to automatically generate ProductionItem baselines of instances for the assembly.

**ProductionItem**

Defined for baselines of instances of hardware which are to be processed using the the **production\_item\_cycle**. When used in conjunction with WordOrder class CRs, baselines of this class are automatically created by the WorkOrder

**Baseline Fields for Standard Project Template**

Various arbitrary fields are defined for baselines in the standard configuration - these may be added to or modified according to requirements.

Many of the fields are used by the **SWRelease** class and **sw\_release\_cycle** to provide information required to implement the release process. The fields are:

Field Name	Class	Description
<b>Pred Baseline:</b>	SWRelease, (None)	defines the predecessor baseline. i.e. the baseline from which this one was created. A value of <b>(Not Applicable)</b> may be used if there is no previous release, otherwise it should contain the name of another release baseline.
<b>Target Release:</b>	SWRelease	defines the target release for CRs which are to be included in this baseline
<b>Rel Dir:</b>	SWRelease	defines the release directory for the product final release.
<b>Rel Workspace:</b>	SWRelease	defines the release workspace in which the product will be built for release purposes. The workspace name must start with 'rel'.
<b>Test Pool</b>	SWRelease	defines the pool which is to be used for testing the release. The pool name must start with 'test'.

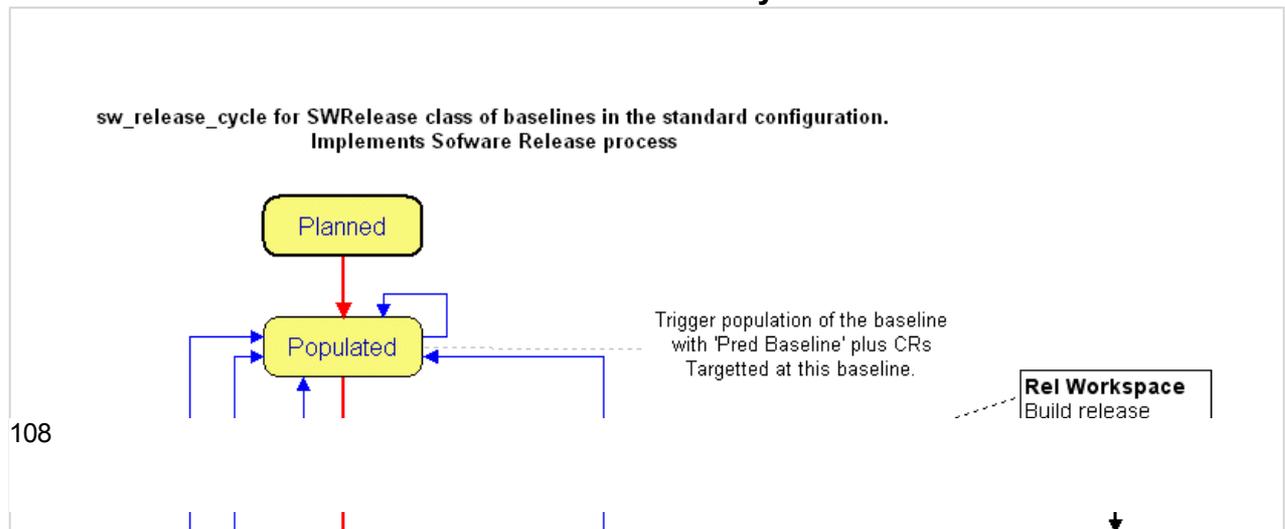
**Baseline Release Management Process for Standard Project Template**

The release management process is implemented by the baseline life-cycles defined for each class of baseline.

**Software Release Management Process**

The software release management process is implemented by the **sw\_release\_cycle** which is associated with the **SWRelease** class of baselines in the standard configuration. The **sw\_release\_cycle** is defined as shown in the [Software Release Cycle](#) diagram:

**Software Release Cycle**



Initially the baseline is in a status of *Planned* which indicates that the release is planned for some future date. The baseline should be created using **Add Baseline with Header Only**.

When the code for the baseline has been developed the baseline should be progressed to *Populated*.

The CRs which are to be released are then prompted for using the **CRs For Baseline** command: CRs which are in a status with the **Release** attribute will be offered for release. SCRs which are in status of Complete and with a **Target Release** of the baseline will be offered (CRs of other classes should not be targeted at baselines of class SWRelease). The baseline details are modified according to the **Pred Baseline** and the code modified to implement the selected CRs. Each CR is associated with the baseline as a *baselinessolved*.

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 workspace specified in the **Rel Workspace** field ready for building; e.g. compile source. For efficiency **Rel Workspace** is actually updated to contain the baseline - i.e. only parts not already checked out and at the correct version will be checked out. Any parts checked out which are not in the baseline are checked in. The **Rel Workspace** is therefore maintained with the baseline in it allowing for incremental releases and modifications to the current baseline without having to re-check out everything.

Once all build processes have been completed and the release is ready for test the status may be progressed to *Test*. If both the **Rel Workspace** and the **Test Pool** fields are defined then all deliverables are promoted from **Rel Workspace** to **Test Pool**.

Deliverables are defined as:

- All files in **Rel Workspace** which match one of the patterns defined in **Global\_Derived\_Deliverables\_Patterns** which is defined in the **Other** configuration options.
- Any files which correspond to parts which have the **Deliverable** field set to **Yes**.

Derived deliverables are moved whereas source deliverables are copied to the **Test Pool**. All CRs included in baseline (i.e. with this baseline as a baseline solved) have their status changed to *Test*.

If it is desired to delete all intermediate object files (\* .obj), then the code for this needs to be uncommented in the **sw\_release\_cycle**.

The *Test* status may not be exited unless all CRs associated with the baseline have been tested (i.e. progressed from *Test* to *Pass* or *Fail*). 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 progressed to *Released*. On release all files will be copied from the **Test Pool** to **Rel Dir** in a flat directory structure. All CRs included in the baseline are also progressed to *Released*.

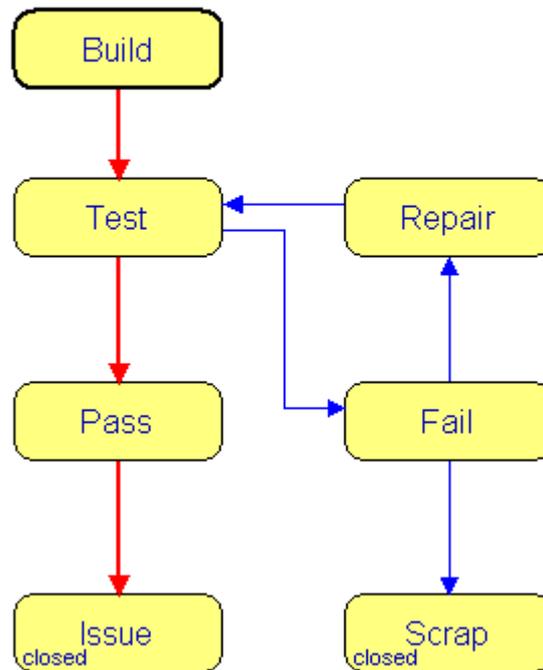
### Production Item Baseline Process

The life cycle for **ProductionItem** class of baselines is defined by the **production\_item\_cycle** in the standard configuration.

The life cycle is defined as shown in the [Production Item Cycle](#) diagram.

### Production Item Cycle

production\_item\_cycle for ProductionItem class of instance release baseline



**ProductionItem** baselines may be created automatically by a **WorkOrder** class CR on entering status **UnderWork**. They will be automatically populated with instances created for each part in the **Assembly** class baseline associated with the **WorkOrder** as a baseline affected.

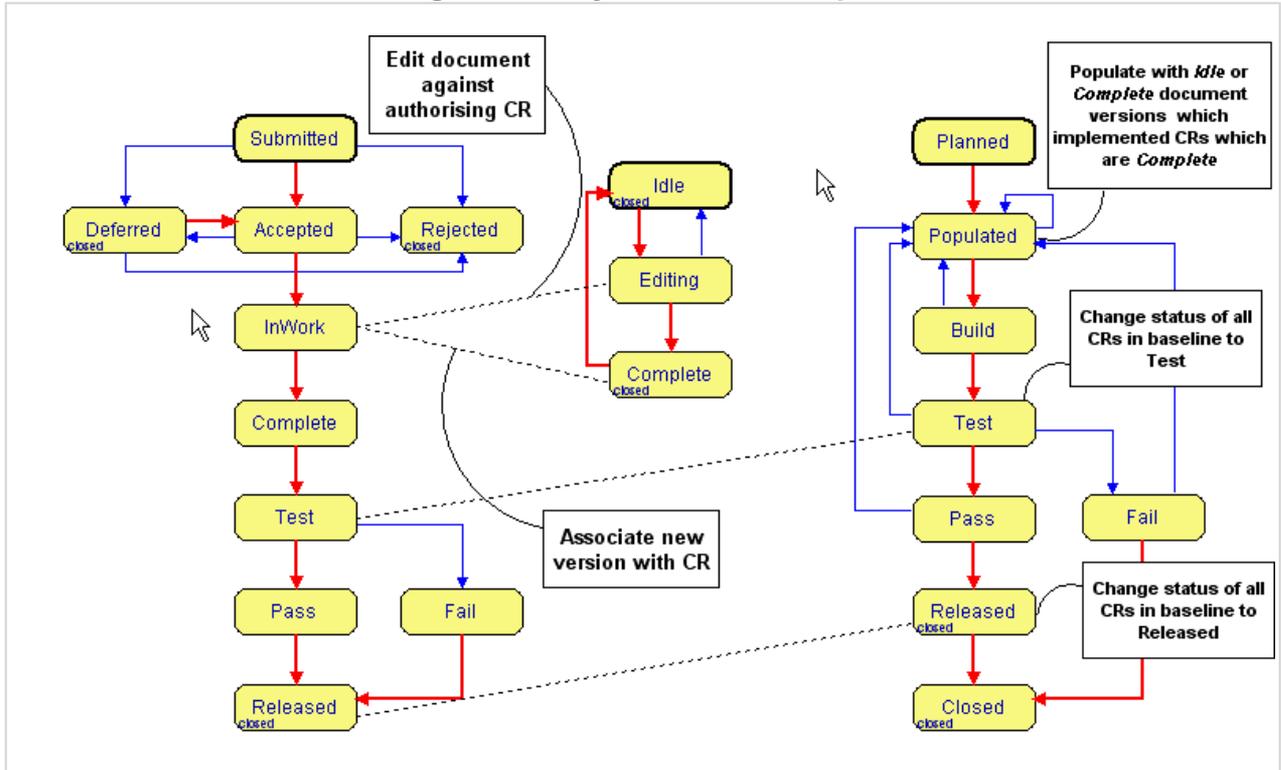
It will start in status *Build* to indicate that the object is being built or manufactured. When ready it is progressed to a status of *Test* to indicate it is being tested and undergoing quality control.

It then either *Passes* or *Fails* testing. If it passes it may then be *Issued*.

If it fails testing then it may either be *Repaired* or *Scrapped*.

Summary of Cycle Relationships when using *sw\_release\_Cycle*

Figure 7.8: Cycle Relationships



**ITIL Configuration**

*About ITIL Project Template*

**AllChange** comes with several ready to use *out-of-the-box* configurations which can be used as they stand or can be tailored to individual requirements.

The ITIL project configuration has been created within the guidelines of the IT Infrastructure Library (ITIL), an industry standard for the Change Management process.

The configuration comprises a set of fully integrated processes from Change Management to Release covering CRs, Parts and Baselines.

The AllChange nomenclature mapping facility has been used in the ITIL configuration to change some of the AllChange standard terminology as follows:

AllChange Standard Term	ITIL Configuration Mapped Term
CR	ACMD - denotes AllChange Change Management Document
Part	CI - denotes Configuration Item

The configuration comprises a set of fully integrated processes from Change Management to Release covering:

**Configuration Items**

- [CI Classes](#)
- [CI Fields](#)
- [CI Change Process](#)

**ACMDs**

- [ACMD Classes](#)
- [ACMD Fields](#)
- [ACMD Change Management Process](#)

#### Baselines

- [Baseline Classes](#)
- [Baseline Fields](#)
- [Baseline Release Management Processes](#)

#### Roles

- [ITIL Roles](#)

#### Reports

- [ITIL Reports](#)

### [Integrated Process Overview](#)

#### *ITIL Parts*

#### CI Classes for ITIL

The ITIL configuration defines the following classes of CI (Configuration Item):

##### **ControlledDoc**

Defined for components which are to be modified by checking in and out using the **review\_cycle** associated with them and are also subject to formal change control, i.e. they may not be created or modified without identifying an ACMD to authorise the change and associate with the change automatically.

##### **ControlledSource**

Defined for components which are to be modified by checking in and out using the **source\_cycle** associated with them and are also subject to formal change control, i.e. they may not be created or modified without identifying an ACMD to authorise the change and associate with the change automatically.

##### **Doors**

Used by the doors interface for check in/out of doors modules. No life-cycle.

##### **HWComponent**

Defined for components which are hardware/manufacturing components with no file associated with them. Components of this class have no life cycle and are the subject of formal change control i.e they may not be created or a new version created without identifying an ACMD to authorise the change and associate with the change automatically.

##### **ITAsset**

Defined for component which are IT Assets to be recorded as an asset register with no file associated with them. Components of this class have the **it\_asset\_cycle** and are the subject of formal change control i.e they may not be created or a new version created without identifying an ACMD to authorise the change and associate with the change automatically.

#### CI Fields for ITIL

The following CI (Configuration Item) arbitrary fields are defined in the ITIL configuration:

##### **Description**

This is defined for all component and subsystem classes of CI. It is used to hold a description of the CI.

##### **Deliverable**

Used to indicate whether the CI is a part of the deliverable product. It is used by the supplied release

process to determine what components to release. This should be used for objects which form part of a product/release and are delivered in the form in which they are stored in **AllChange** - i.e they do not undergo a transformation process and the derived object is what is released (e.g source → compiled object)

#### **Reviewer**

This is defined for components of class **ControlledDoc**. It should be used to specify the name of the user who is to be the document reviewer. This is compulsory and will be prompted for on creation of the part.

#### **No Doc Stamp**

This is defined for components of class **ControlledDoc**. If ticked the document will not be checked for any AllChange fields to be stamped. This should be used for all components Word or Excel documents where it is known that document stamping is not used in order to improve performance.

#### **Manufacturer**

This is defined for parts of class **ITAsset** and may be used to hold who the manufacturer of the IT asset is.

#### **Supplier**

This is defined for components of class **HWComponent** and **ITAsset** and may be used to hold the who the supplier of the item is.

#### **Location**

This is defined for versions of components of class **ITAsset**. This may be used to store the physical location of the asset.

#### **Comment**

This is defined for versions of components with no class which do not require a CR in order to be checked out for edit. This is compulsory and will be prompted for on check in.

#### **Notes**

This is defined for versions of components of class **ITAsset** and **HWComponent**. This is compulsory and will be prompted for on creating a new version.

#### **ArchType**

This is defined for versions for all classes which have a file and indicates the type of archiving. This may be OnLine, OffLine or empty.

#### **Merge From**

This is defined for versions for components of class **ControlledSource** and is used to hold information as to which versions this version was merged from. This is set automatically when using the All-Change Merge facility

### **CI Change Processes for ITIL**

For all components they may be modified using one of two methods:

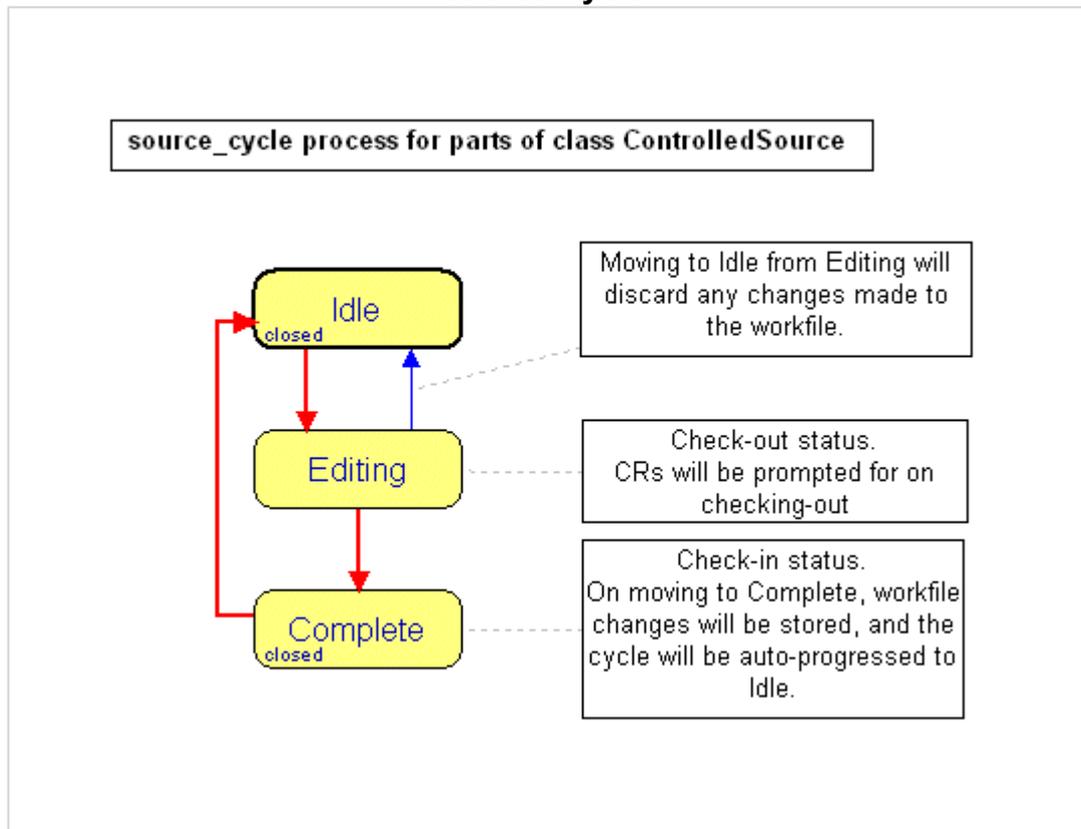
- **Check Out/Check In** uses issue/return commands directly
- **Check Out/Check In** makes use of the life-cycle to do the issue/return commands for you

Which of these methods is used is determined by whether there is a life-cycle associated with a particular CI class .

#### **ControlledSource Cycle Driven Change Process**

The **ControlledSource** class of component is associated with the **source\_cycle** life-cycle, this is shown in the Source Cycle diagram below.

## Source Cycle



In order to make a change the required version should be **checked out**. The **Check Out** command will change the status of the version from *Idle* to *Editing*, this will create a new version and check the version out for edit into the current workspace. This may be achieved by using **Check Out** or double clicking on the Editing status in the Cycle View window. *Check Out statuses should not be set by changing the status explicitly using the Status button on the Part Viewer.*

If the component is of class **ControlledSource** then the ACMDs which authorise the change must be specified. The classes of component which are subject to formal change control in this way is specified as an attribute of the class.

The ACMDs will be prompted for by **Check Out**.

Only those ACMDs which are valid for authorising a change will be shown. This will include ACMDs which are assigned to you and are of class **RFC** in a status of *InAction* or *Approved*. The class of the ACMDs and the status that they should have to authorise the change is defined in the ACMD life-cycle definition.

The ACMD numbers authorising the change will be stored in the check-out record's comment field. Also, the CIs checked out will be added to the ACMDs' partsaffected. This means that it is possible to see which CIs are, or have been, checked out against an ACMD by looking at the partsaffected, unless CIs have been manually removed from the ACMD's partsaffected.

When you have finished making your changes and wish to check your code/document back in either use **Check In** which will progress the status to *Complete* or double click on the *Complete* status in the Cycle View Window. This will return the component from edit, storing the new version under control.

If the component was checked out against once or more ACMDs, it will also associate the new version with the ACMDs against which it was edited as both partsaffected by each ACMD if it is not already one and the new version as the versionsolved by each ACMD. If a new version was not checked-in, or if the edit was discarded, then the component is removed as a partsaffected providing there is not already a version solved for the part. If the **Only one version solved** configuration option is selected then any existing

version in a ACMD's versionsolved will be replaced with the new version. If **Only one version solved** has not been set then the new version will be added to the ACMDs' versionsolved.

If a component version is checked out for edit (the status of a component version is *Editing*) and you wish to discard any changes you have made and effectively change your mind about modifying the component then **Check In** may be used and select the **Discard** option or the status may be changed from *Editing* back to *Idle* and this will effectively reverse the status change to *Editing* as though it had never happened. When a version has been *Completed* its status will be automatically set back to *Idle* ready for further changes.

### ControlledDoc Cycle Driven Change Process

The **ControlledDoc** class of component is associated with the [review-cycle](#).

This cycle is intended for CIs that need to undergo a review procedure, that is, all changes to the CI must be approved by a reviewer.

The **Reviewer** for the document is specified on an arbitrary field named Reviewer.

When a new document is checked in the user is prompted to select whether the first version of the document is a *Draft* or is *Approved*.

Draft versions may be checked out for modification (this will enter the *Editing* status) or progressed to the *Review* state indicating that the document version is ready for review. On entry to Review the reviewer will be mailed to inform him/her that the document is ready to be reviewed.

On check out for edit the ACMDs authorising the change will be prompted for by **Check Out**.

Only those ACMDs which are valid for authorising a change will be shown. This will include ACMDs which are assigned to you and are of class **RFC** in a status of *InAction* or *Approved*. The class of the ACMDs and the status that they should have to authorise the change is defined in the ACMD life-cycle definition.

The ACMD numbers authorising the change will be stored in the check-out record's comment field.

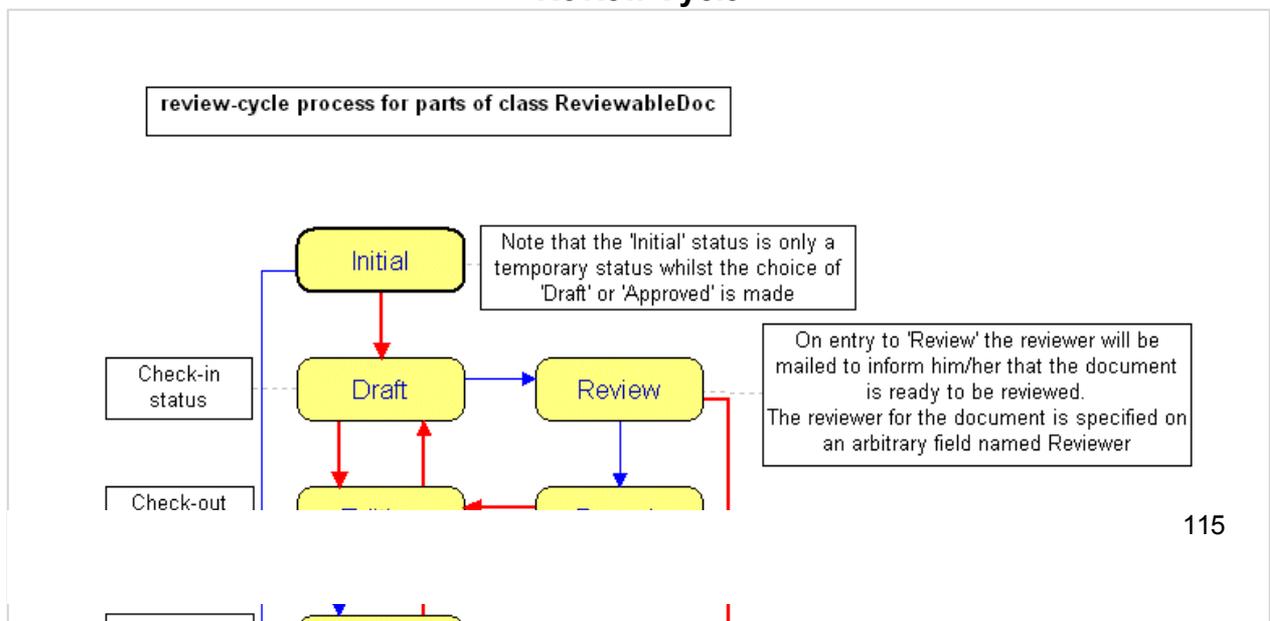
When a checked out document is checked back in a choice may be made as to whether the new version is a *Draft* or is *Approved*. Note that only the document reviewer may check it in as approved.

The new version will be associated with the ACMDs against which it was edited as both parts affected by each ACMD if it is not already one and the new version as the versionsolved by each ACMD. If the **Only one version solved** configuration option is selected then any existing version in a ACMD's versionsolved will be replaced with the new version. If **Only one version solved** has not been set then the new version will be added to the ACMDs' versionsolved.

When a reviewer has reviewed the document it may be either *Approved* or set to *Rework*. If Rework is required then the author will be emailed to inform them that further work is required. The document may then be checked out for editing and modifications made.

In the diagram below the life-cycle is shown. Note that the Initial status is only a temporary status whilst the choice of *Draft* or *Approved* is made.

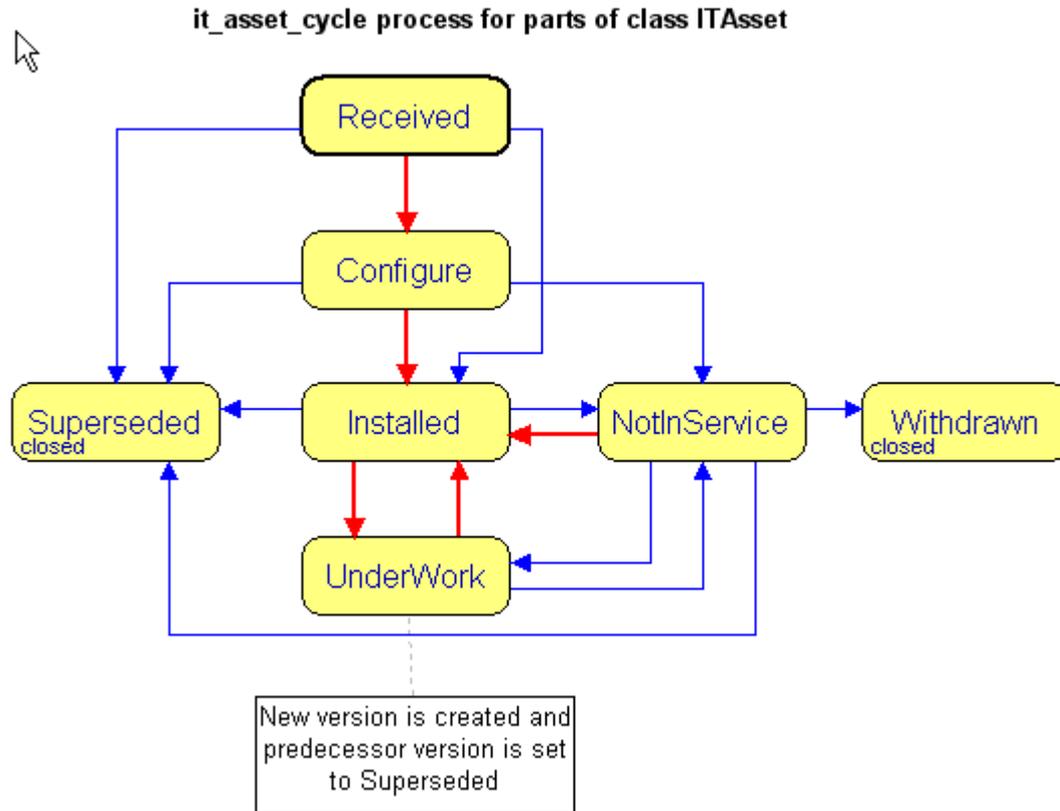
### Review Cycle



### ITAsset Cycle Driven Change Process

The **ITAsset** class of component is associated with the [it\\_asset\\_cycle](#) life-cycle shown below.

### IT Asset Cycle



This process is for use when using **AllChange** to store information about IT Assets as CIs with the **No File** flag. It allows the process of an asset being received through to becoming withdrawn or no longer used to be tracked.

Initially the asset is *Received*, it may then require some configuration, in which case it can be progressed to *Configure*.

Having been configured or if no configuration is needed it may then be progressed to *Installed* to indicate it is now up and running and being used.

Once installed the asset may then go through a number of changes (e.g. A PC may have software installed on it, its memory upgraded etc). This may be recorded through the *UnderWork* status which will cause a new version to be created and the predecessor version to have its status set to *Superseded*. In order to enter *UnderWork* an ACMD must be specified as the authority for the change if the part is of class **ITAsset** or any other class which requires an ACMD to authorise a change. When the work is complete it may then be set to *Installed* to denote that the new version is up and running and being used.

At any time when the asset is *Installed* it may be set to *NotInService* to denote that it is no longer in use but is still available. *NotInService* assets may be brought back into service by setting their status to *Installed*, or may be *Withdrawn* entirely

#### Change Process with no Life-Cycle

If you do not wish your change process to be cycle driven then you may use the classes which do not have a life-cycle associated with them and the **Check Out** and **Check In** will use issue and return commands directly instead.

When you wish to check out a part for edit you should use **Check Out** and when you have completed your changes you should **Check In**. Components of class **doors**, for example, will be modified in this way.

In the ITIL configuration these classes also do not require an ACMD to authorise their change.

## ITIL CRs

### ACMD Classes (ITIL)

In the ITIL configuration the AllChange term CR has been redefined to the term ACMD (AllChange Change Management Document)

The ITIL configuration defines the following classes of ACMDs.

#### Incident

**Incidents** are raised as a result of a service desk call. **Incidents** are associated with the **ITIL\_Incident\_Cycle** life cycle. ACMDs of class incident are numbered with a IR prefix followed by a five digit auto incrementing number, e.g. IR00001.

#### Problem

**Problems** are used to record known problems and errors. A Problem may be raised as the result of an Incident or may be raised independently. **Problems** are associated with the **ITIL\_Problem\_Cycle**. ACMDs of class problem are numbered with a PR prefix followed by an auto incrementing number, e.g. PR00001.

#### RFC

**RFCs** are *requests for change* which may result from a **Problem** or may be raised independently. They authorise actual component changes. **RFCs** are associated with the **ITIL\_RFC\_Cycle** and are numbered with a RFC prefix followed by a five digit auto incrementing number, e.g. RFC00001.

#### WorkOrder

Defined for ACMDs for manufacturing, to be used with baselines of class Assembly and ProductionItem. WorkOrders are associated with the wo\_cycle life-cycle. ACMDs of class WorkOrder are numbered with a WO prefix followed by a five digit auto incrementing number e.g. WO00001

These classes illustrate the 3 hierarchical levels of change/problem management within ITIL.

### ACMD Fields (ITIL)

Various arbitrary fields are defined for all classes of ACMD in the ITIL configuration - these may be class specific and modified according to requirements. There are also many more arbitrary fields available which may be defined as required. The Fields defined include those needed to store the information recommended by ITIL and which allow automation of mailing and access control for the ITIL Change Management Process.

Certain fields are common to all classes of ACMD, these include:

**Locked:** Whether the ACMD is locked against any further change

All classes also have defined the following fields on the CR items affected:

**Description:** A description of the relationship

**Obsolete:** Whether the relationship was created due to the item being made obsolete. This is set to Yes when a controlled part is made obsolete against an ACMD.

**Incident Fields**

<b>Incident Urgency:</b>	indicates the time required for the incident to be resolved, values are: <b>Immediate</b> <b>24 hours</b> <b>48 hours</b> <b>Low</b>
<b>Impact:</b>	indicates the impact of the incident, values are: <b>Critical</b> <b>High</b> <b>Medium</b> <b>Low</b>
<b>Raised By:</b>	The person who reported the incident
<b>Phone Number:</b>	Contact phone number for the person who reported the incident
<b>Type:</b>	Categorises the incident, values are: <b>Application</b> <b>Hardware</b> <b>Service Request</b>
<b>Incident Priority:</b>	Denotes the priority that the Incident has, values are: <b>Immediate</b> <b>High</b> <b>Medium</b> <b>Low</b>
<b>Company:</b>	The name of the company reporting the incident
<b>Email:</b>	Email address for user who reported the incident
<b>No Problems:</b>	This indicates the incident is not a recorded Problem.

In addition the Incident Text tab has the following sections:

Incident Description	This should contain a detailed description of the incident
Investigation Notes	This should contain details of steps taken to identify the potential scope and severity of the incident
Resolution	This should contain details of action taken, including the creation of RFCs, to resolve the incident

**Problem Fields**

The standard fields for the Problem class are:

<b>Known Error:</b>	Indicates this problem has been logged before and is a known error
<b>Classification:</b>	Allows a the problem to be classified as required
<b>Problem Priority:</b>	Denotes the priority that the Problem has, values are: <b>Immediate</b> <b>High</b> <b>Medium</b> <b>Low</b>

In addition the Problem Text tab has the following sections:

Description of Problem	This should contain a detailed description of the problem
Resolution	This should contain details of how and action taken, including the creation of RFCs, to resolve the problem

**RFC Fields**

<i>Actual Implementation Date</i>	The Actual date of implementation of the change as opposed to Scheduled Implementation Date.
<i>Affects Capacity Plan</i>	Whether this change affects the Capacity Plan. Should be used in Impact Assessment.
<i>Affects ITSCM Plan</i>	Whether this change affects the IT Service Continuity Management (ITSCM) Plan. Should be used in Impact Assessment.
<i>Affects Security Plan</i>	Whether this change affects the Security Plan. Should be used in Impact Assessment.
<i>Affects Test Plan</i>	Whether this change affects the Test Plan. Should be used in Impact Assessment.
<i>Assessors</i>	The people responsible for assessing and evaluating the RFC. These should be allocated by Change Management during initial Review.
<i>Business Executive</i>	Name of people in the Business Executive required for authorisation of the RFC and as members of the Change Authority. These should be allocated during assessment and evaluation by the assessors if members of the Business Executive are required for authorisation of this change.
<i>CAB</i>	Members of the global CAB who are to act as the Change Advisory Board for the RFC and as members of the Change Authority. These should be allocated during assessment and evaluation by the assessors if members of the CAB are required for authorisation of this change. For a very minor change, for example, the CAB may not be required and local authorisation may be sufficient.
<i>Change Category</i>	The category indicates whether the change is Major, Minor or Significant. This may be set by the Originator but may be reassessed during assessment and evaluation.
<i>Change Cause</i>	This should indicate the cause of the change, for example Business Need, Purchase Order, Error Record, Legislation, Problem or Service Request. Where relevant (e.g. Problem, Purchase Order) the related Problem or Incident should be associated with the RFC or other reference may be specified in the Reference field. This should be set by the originator of the change and is mandatory on creation of an RFC
<i>Change Type</i>	Level of change — <b>Normal</b> , <b>Standard</b> or <b>Urgent</b> . This determines the change process to be used, i.e. it determines the route through the life-cycle. This must be set by the originator on creation of the RFC
<i>ECAB</i>	Members of the global CAB who are to act as the Emergency Change Advisory Board for Emergency changes. The ECAB may be a subset of the full CAB in order to progress emergency changes through quickly.
<i>Estimated Effort</i>	This may be used to record the estimated effort to implement the change. This may be useful in assessing the cost of implementing the change. This should be completed as a part of the impact assessment and risk evaluation.
<i>Hardware Costs</i>	This may be used to record the costs of any hardware required to implement the change. This should be completed as a part of the impact assessment and risk evaluation.
<i>Impact Level</i>	This should denote the level of impact of the change as High or Low and should be completed during assessment and evaluation as a part of the impact assessment and risk categorization. This

	together with the <b>Probability</b> of failure will be used to calculate the <b>Risk Category</b> .
<i>Implementors</i>	This should be the members of the Deployment role who will be responsible for implementing the change. This should be allocated during assessment and evaluation as a part of resource allocation.
<i>IT Management</i>	Name of people in IT Management required for authorisation of the RFC and as members of the Change Authority. These should be allocated during assessment and evaluation by the assessors if members of IT Management are required for authorisation of this change.
<i>Priority</i>	The priority of the change as Low, Medium, High or Immediate. Immediate should be used for Emergency changes. This is set during assessment and evaluation of the change. This should be derived from the agreed impact and urgency of the change.
<i>Probability</i>	This should denote the probability of the change failing as Low or High and should be completed during assessment and evaluation as a part of the impact assessment and risk categorization. This together with the <b>Impact Level</b> will be used to calculate the <b>Risk Category</b> .
<i>PSO</i>	The Projected Service Outage (PSO) should the amount of time a service is expected to be unavailable whilst the change is being implemented and should be completed during assessment and evaluation as a part of the impact assessment and risk categorization.
<i>Release Type</i>	The Release Type specifies the type of release that this change will be included in. For Software and Documentation Release Types the RFC process is integrated with the Baseline Release Process.
<i>Requester Email</i>	Email address of person requesting the change.
<i>Requester Name</i>	Name of person requesting the change. This should be completed in all cases where the originator of the RFC is not the same as the requestor.
<i>Requester Phone</i>	Phone number of person requesting the change.
<i>Resource Costs</i>	This may be used to record the costs of any resources required to implement the change. The allocated Technical and Test Teams and the Implementors together with the Estimated effort may be useful in assessing this cost. This should be completed as a part of the impact assessment and risk evaluation.
<i>Risk Category</i>	The category/level of risk of implementing this change. This is calculated automatically from the <b>Impact Level</b> and <b>Priority</b> according to a simple risk categorization matrix.  High impact + High probability = Risk Category 1 High impact + Low probability = Risk Category 2 Low impact + High probability = Risk Category 3 Low impact + Low probability = Risk Category 4
<i>Scheduled Completion Date</i>	Date of expected completion of the work to effect the change.
<i>Scheduled Implementation Date</i>	Date of expected implementation.
<i>Software Costs</i>	This may be used to record the costs of any software required to implement the change. This should be completed as a part of the impact assessment and risk evaluation.
<i>Target Release</i>	The Target Release is used to specify the baseline in which this RFC is intended to be included. This is used for Software and

<i>Technical Team</i>	Documentation Release Types. Team for building/effecting the change. This may be selected from a list of users with the Application Developer and Builder roles.
<i>Test Team</i>	Team for testing. This may be selected from a list of users with the Tester role.
<i>Total Costs</i>	This is automatically calculated as the total of the Software, Hardware and Resource costs.
<i>Urgency</i>	The originator/requestors perceived urgency of the change. This is based on how long the implementation of the change can be delayed.

The standard fields for the RFC class are:

In addition the RFC Text tab has the following sections:

<i>Description of Change</i>	This should contain a detailed description of the requested change
<i>Reason for Change</i>	This should contain a detailed description of the reason for requesting the change
<i>Effect of Non Implementation</i>	This should contain a describe the effect on the business if the change is not implemented.
<i>Impact Assessment and Evaluation</i>	This should contain a detailed description of the impact assessment of the change - in addition a document may be attached to the RFC detailing this information and the text here can refer to that.
<i>Risk Assessment and Management Plan</i>	This should contain a detailed description of the risk assessment of the change and detail how this risk is to be managed - in addition a document may be attached to the RFC detailing this information and the text here can refer to that.
<i>Implementation Plans</i>	This should contain details of how the change is to be implemented. If required documents relating to the implementation plans may also be attached to the RFC and then referenced in this text section.
<i>Back-out Plans</i>	Some Implementation Plans must be specified before the RFC can be submitted to the CAB for review This should contain details of how the change can be backed out if implementation fails for any reason. If required documents relating to the backout plans may also be attached to the RFC and then referenced in this text section
<i>Change Authority Comments</i>	Some Implementation Plans must be specified before the RFC can be submitted to the CAB for review This should contain any comments from the Change Authority if required.
<i>PIR Comments</i>	This should contain any comments from the post implementation change review prior to Closure of the RFC
<i>Reject Reason</i>	The reason for rejection of the RFC

#### **WorkOrder Fields**

The standard fields for the **WorkOrder** class are:

<b>Quantity</b>	The number of items to be manufactured.
<b>Quality Control</b>	Whether the items manufactured as a result of this WorkOrder are subject to Quality Control.

In addition the RFC Text tab has the following sections:

Description of Work Required	This should contain a detailed description of the requested Work Order
Operational Impact	This should detail any operational impact of undertaking the WorkOrder
Identify any Risk Areas	This should contain details of any risks associated with undertaking or not undertaking the WorkOrder
Cost Implications	This should contain any costing information

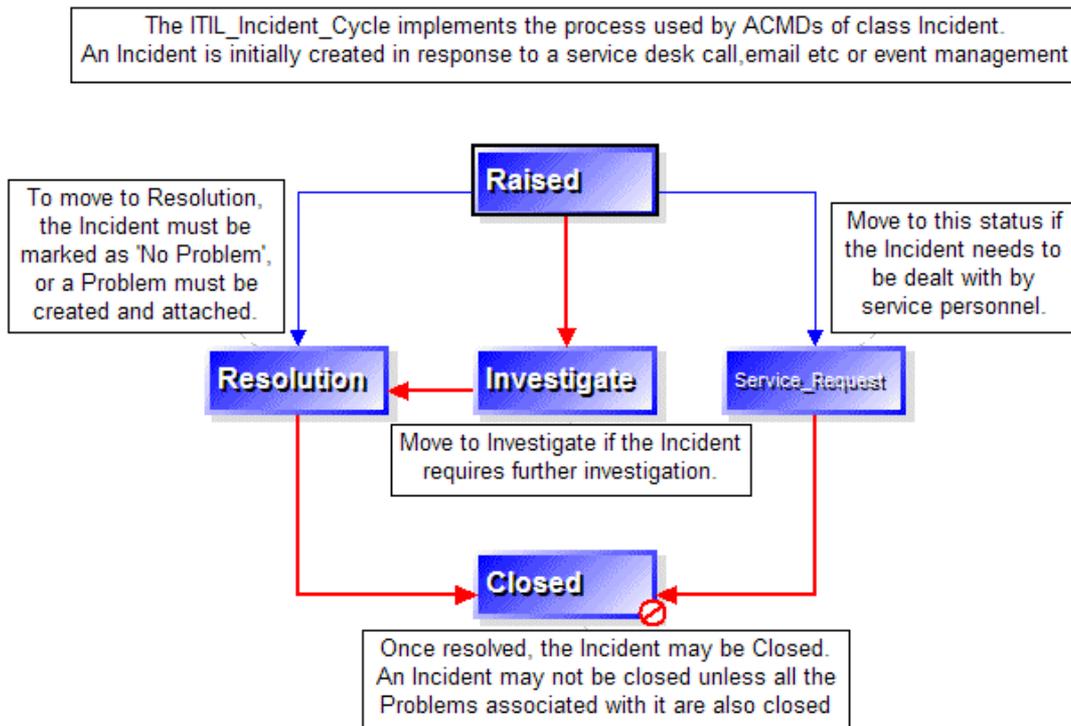
**ACMD Change Management Process (ITIL)**

The change management process is implemented by the CR life-cycles defined for each class of CR. In the ITIL configuration CRs have been renamed ACMD's  
 There are three classes of ACMD in the ITIL configuration.

- An **Incident** is initially created in response to a service desk call
- If the Incident determines that there is a Problem, a **Problem** record is created
- An **RFC** (Request For Change) is created either in response to a Problem or in its own right.

**Incident Management Process**

The process implemented by ACMDs of class **Incident** is determined by the ITIL\_Incident\_Cycle shown below.



When an Incident is *Raised* a **Summary** must be given to summarise the Incident and a **Priority** must be specified. Other information may be supplied if it is available, in particular a Description of the incident

should be made in the ACMD Text. You will also be prompted to select a user to assign the Incident to, select Cancel if it is not to be assigned.

If the Incident needs to be dealt with by service personnel it should be set to *Service Request* and assigned to an appropriate person. Once the incident has been dealt with the status should be set to *Closed*.

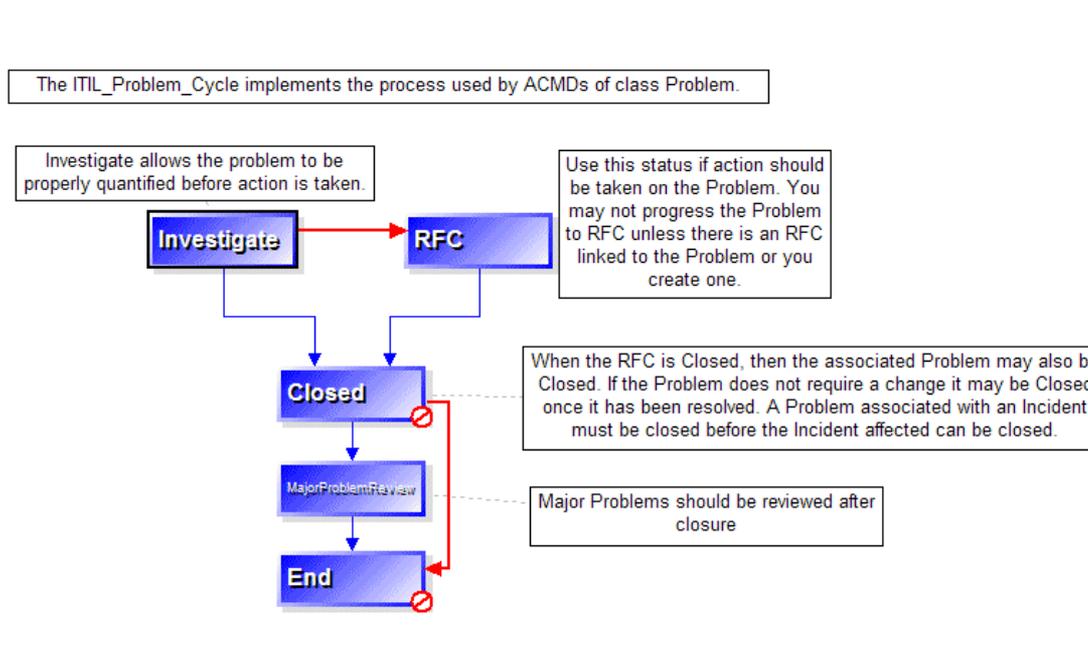
If the Incident can be effectively dealt with immediately then it may be progressed to *Resolution*, and if necessary assigned to an appropriate person and once resolved may be *Closed*.

If the Incident requires further investigation, it should be set to *Investigate* and assigned to an appropriate person. The ACMD should be updated with details of the investigation, including **Type** and **Investigation Notes**. If it is established as not being a problem the **No Problem** should be checked. Otherwise if it is determined to be a problem it should be linked to an existing problem if there is already one or a new one should be created and linked to the Incident. **ACMD | Link Incident to Problem** may be used to link an incident to an existing problem. If a new problem needs to be created then **ACMD | Create Problem from Incident** may be used to create the problem and link it to the incident. It may then be progressed to *Resolution*.

Once resolved the Incident may be *Closed*. An Incident may not be closed unless all the Problems associated with it are also closed

### Problem Management Process

The process implemented by ACMDs of class **Problem** is determined by the ITIL\_Problem\_Cycle shown below.



When a Problem is created the only mandatory field is **Summary**, however it is recommended that a Description of the Problem is entered in the Text as well. On creation you will be given the opportunity to link the problem to an Incident.

From an initial status of *Investigate*, allowing the problem to be properly quantified before action is taken it can be progressed to *RFC*, indicating a change is required (This may require the creation of an **RFC**.)

If the Problem requires a change to resolve it you may create one or more RFCs to implement the change and link the Problem to them, or on entry to RFC you will be given the opportunity to create an RFC at that time. You may not progress the Problem to RFC unless there is an RFC linked to the Problem or you create one. On progressing from Investigate to RFC if there is not already an RFC associated with the problem you will be prompted to create one which will then be linked to the problem. Alternatively an RFC

may be created and linked to the problem using **ACMD | Create RFC from Problem**, or it may be linked to an existing RFC using **ACMD | Link Problem to RFC**. On entry to RFC status the **Known Error** field will be automatically set to **Yes**.

When the RFC is *Closed* then the associated Problem may also be *Closed*.

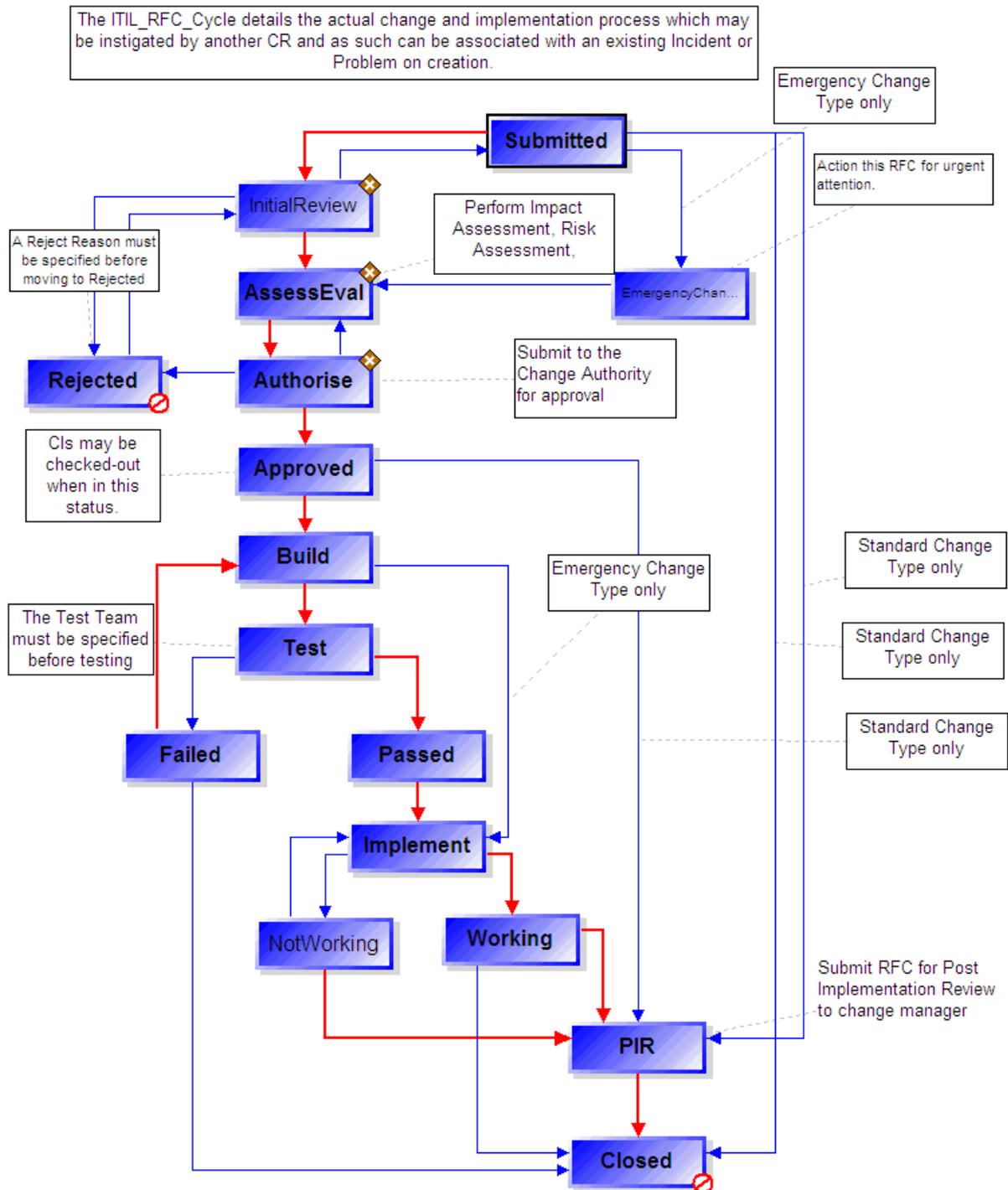
If the Problem does not require a change it may be *Closed* once it has been resolved.

Any Major problems must go through a review (*MajorProblemReview*) after the problem has been closed.

A Problem associated with an **Incident** must be closed before the Incident affected can be closed.

### **Change Management Process**

RFCs detail the actual change and implementation process which may be instigated by an Incident or Problem, if it is then it must be associated with an existing Incident or Problem on creation.



Initially the RFC is **Submitted**, its Type should be specified together with the Cause, Urgency and a Summary of the change. A Description of the change should be entered together with Requester details and any other information available if appropriate.

According to the Type allocated, 3 different change processes may be required:

1. **Normal**: this is the usual change management process which involves the RFC being reviewed and if accepted traced through the process of development through to implementation
2. **Standard**: this is the process used for changes which are regarded as standard and are pre-approved (by a previous Normal RFC). This should be used for changes which occur frequently, where the implementation and backout plans are well established and repeatable and where full

tracking of the change is too much of an overhead, but traceability that the change has occurred is desired. An example of a Standard type of change might be addition of a new user login to the network, acquisition and build of a new standard spec PC. A Standard RFC should refer to its authorising RFC (i.e. the RFC that determined this type of change is to be regarded as Standard).

Standard authorising RFCs should be reviewed on a regular basis to review whether their Standard categorisation should be revoked. Post implementation **Reviews** of Standard RFCs should be useful in this determination

- 3. Emergency:** this is the process used for changes which are regarded as urgent/emergencies, e.g. may be causing interruption of normal business activity, where a fast track through the change process is needed in order to resume business as usual as fast as possible. It may also be used to record retrospectively that a change has occurred, e.g. A Change had to be made urgently to the live system when no access to the change management system is available, in this case as soon as possible after the change an RFC should be raised to record the details of the change.

The type of change process is referred to as the **Change Type** and is specified in a field of the RFC

The process for each of these change categories is described below in terms of the statuses in the life-cycle

## Normal Change Process

<i>Submitted</i>	An RFC of Type Normal may be submitted for <i>InitialReview</i> by the Change Manager once the Type and Cause and full details of the requested change have been entered. If required a full change proposal may be attached to the RFC. The RFC may be raised by the initiator or by someone else on their behalf, in this case the Requester Details should be completed. An RFC may also be returned to the <i>Submitted</i> status by Change Management after initial review, for example due to incomplete information. The RFC may then be resubmitted for review.
<i>InitialReview</i>	An RFC of Type Normal may be submitted for initial <i>Review</i> by the Change Manager once the Type and Cause and full details of the requested change have been entered. On entry to the <i>InitialReview</i> status the RFC will be automatically assigned to Change Management. Change Management should briefly review the RFC and any which are viewed as totally impractical, repeats of earlier RFCs or incomplete submissions should be returned to the originator by failing the review. During initial review a vote must be cast by one of the Change Managers before the RFC may be progressed. If the RFC is deemed to be suitable for full assessment and evaluation then the users who are to perform the assessment and evaluation should be nominated in the Assessors field on the Resources Tab. In order to progress the RFC a vote is required by one of the change management team to either submit it for assessment and evaluation, <b>AssessEval</b> , or return it to the originator for additional information, <b>Submitted</b> . If returning to the originator then a comment should be recorded on the vote as to the reason for the failure. On setting the status to <i>Submitted</i> the RFC will be automatically re-assigned to the originator. On setting the status to <i>AssessEval</i> the RFC will be assigned to the nominated assessors.
<i>AssessEval</i>	<p>On entry to the <i>AssessEval</i> status, the RFC is automatically assigned to the nominated <b>Assessors</b> for this RFC, this will cause an email to be sent to each Assessor informing them of this. During assessment and evaluation the following should be considered:</p> <ul style="list-style-type: none"> <li>Who RAISED the change?</li> <li>What is the REASON for the change?</li> <li>What is the RETURN required from the change?</li> <li>What are the RISKS involved in the change?</li> <li>What RESOURCES are required to deliver the change?</li> <li>Who is RESPONSIBLE for the build, test and implementation of the change?</li> <li>What is the RELATIONSHIP between this change and other changes?</li> </ul> <p>Details to answer all the above may be recorded on the RFC some of which should have been completed prior to submission for assessment and evaluation (Who Raised, the Reason and the Return). See <a href="#">Fields Defined for RFCs</a> for details of all data fields in the supplied ITIL configuration, but this information may be recorded on the Change Details, Impact/Risk Assessment, Resources, Costs, Schedule and Text tabs. During assessment and evaluation the Assessors should consider and record the Impact of the Change, the level of impact and probability of failure should be assessed in order for a risk category to be determined. A full impact assessment should be performed and recorded (impact assessment forms/documents may be attached to the RFC), also the change should be planned/scheduled. The cost of making the change should be con-</p>

Change Authority for approval of the change, this will require the CAB and possibly IT or Business Executive. These may all be nominated on the Change Authority tab. The resources required to make the change in terms of the technical team who will perform the work, the testers who will test the change and the implementor who will implement the change can all be recorded on the Resources tab.

The Release Type must be specified as either Infrastructure, Manufacturing, Software or Documentation, this is used in the release process.

The configuration items affected by the RFC can be associated with the RFC using the Relationships tab as can baselines affected.

The assessors may record their decision on the RFC by casting a vote and specifying a comment.

*Authorise*

Once assessed and evaluated the RFC may be progressed for authorisation by the Change Authority by progressing the RFC to the *Authorise* status.

The RFC has been submitted to the Change Authority for authorisation. This will open a vote which will cause an email will be sent to inform those individuals. The Change Authority should review the information on the RFC and may review the votes cast by the Assessors in order to make a determination for the change. Each member of the Change Authority should cast a vote for **Rejected**, **Approved** or **AssessEval** (to return the RFC to the assessors for further assessment, a unanimous decision is required. Reassessment should be performed if it is deemed that insufficient assessment an evaluation has been performed and further is needed before a decision can be reached. If there is no unanimous decision then the RFC should be assigned to a higher level of authority (e.g. IT Management or Business Executive) for a final decision to be made. If the RFC is to be approved then the Technical, Test and Implementation teams should be allocated on the Resources tab.

*Rejected*

The RFC has been rejected by the Change Authority. A **Reject Reason** must be specified before an RFC may be rejected.

*Approved*

The change has been approved by the Change Authority. On approval the **Technical Team** allocated will be mailed to inform them that the RFC is approved and may be 'done'/developed. If no Technical Team has been allocated some appropriate authority (e.g. Team Leader) should be assigned the RFC and should allocate the **Technical Team**. In this status CI's may be changed against this RFC and will be automatically associated with the RFC as CIs Affected and Versions solved by the change.

If the Release Type is Software or Documentation then a Target Release must be specified before progressing to the Build status. When the work to do the change has been completed the status should be progressed to Build.

*Build*

The work on the RFC has been completed and is now ready for release. For Release Type Software or Documentation, when all RFCs for the Target Release are in the Build status, the release may be progressed using the baseline specified. RFCs in a status of *Build* with the Target Release are now valid for population into the baseline. Once the work is completed the change should be tested.

*Test*

The work detailed in the RFC is under test and may either **Pass** or **Fail**. On entry

to **Test** the members of the **Test Team** and they are mailed to inform them that this RFC is ready for testing. For RFCs with a Release Type of Software or Documentation the RFC is progressed to the test status automatically by the baseline in which the RFC is included for release.

<i>Fail</i>	The work detailed in the RFC has failed the testing stage. It may return to <b>Build</b> for further work or the RFC may be <b>Closed</b> . On entry to Fail the <b>Change Manager</b> is mailed to inform them of this.
<i>Passed</i>	The work detailed in the RFC has passed the testing stage and may be implemented. On entry to <b>Pass</b> the <b>Change Manager</b> is mailed to inform them of this.
<i>Implement</i>	The work detailed in the RFC is ready for implementation. The designated Implementors will be emailed to inform them of this. For RFCs with a Release Type of Software or Documentation the RFC is progressed to the test status automatically by the baseline in which the RFC is included for release. From here the work will be found to be <b>Working</b> or <b>Not Working</b> .
<i>Not Working</i>	The changes have been applied and have not been successful. A further attempt to implement the change may be made or the Back Out Plans may be invoked and the RFC then needs to go for a Post Implementation Review ( <i>PIR</i> ) to assess the reasons for failure prior to being <b>Closed</b> .
<i>Working</i>	The changes have been successfully implemented. The RFC can go for a Post Implementation Review ( <i>PIR</i> ) after an agreed period of time or be <b>Closed</b> .
<i>PIR</i>	An assessment of the effectiveness of the change. On entry to this status the RFC is automatically assigned to the Change Manager who will then be emailed to inform them this RFC is ready for post implementation review. As a part of the Review Some <b>Post Implementation Review Comments</b> must be made, it may then be <b>Closed</b> .
<i>Closed</i>	The work detailed in the RFC has been completed and this RFC has been ended.

### Emergency Change Process

The Emergency change process follows the normal change process except that:

1. Documentation of the change may be made retrospectively - for example where a change had to be made out of normal hours in order to prevent interruption to an important business service
2. In exceptional cases testing may be reduced or in extreme cases omitted all together.
3. An Emergency CAB (ECAB) may be used to authorise the change instead of the full CAB

Thus in general documentation of the Normal change process above should be referred to for an emergency change with the following exceptions:

<i>Submitted (Initial Status)</i>	The RFC has been submitted for approval. If the changes described in the RFC have been accepted as emergency the <b>Type</b> will be <b>Emergency</b> . The RFC may move to <b>EmergencyChange</b> or be moved through the normal change process.
<i>EmergencyChange</i>	This denotes the building and testing of the changes described in the RFC. This may be entered retrospectively. Once complete the RFC may be progressed to <b>AssessEval</b> for assessment and eval-

uation.

*AssessEval*

The RFC is to be assessed and evaluated. For an Emergency change this may involve the ECAB rather than a full CAB being allocated as the Change Authority. For an Emergency Change the resources do not need to be allocated if this is a retrospective RFC created after the work has already been performed.

**Standard Change Process**

*Submitted (Initial Status)*

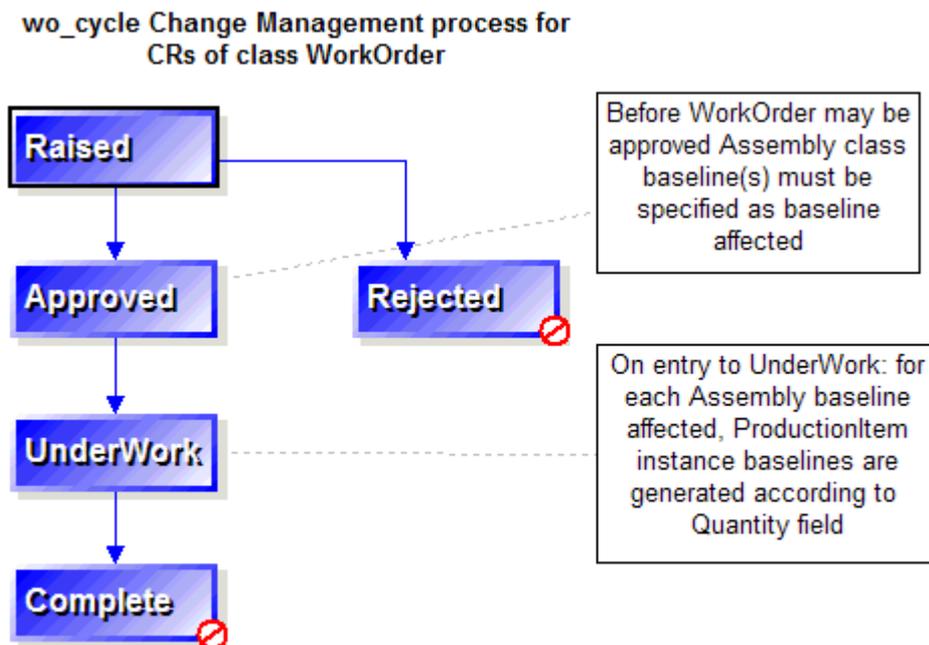
The RFC has been submitted and the Type assessed as Standard. It may then be **Closed** immediately or submitted for **Post Implementation Review (PIR)** once the change has been completed. Alternatively for certain types of standard change it may be desirable to undergo the normal change authorisation process, but once Approved, simply be submitted for Post Implementation Review.

*Approved*

The work detailed in the RFC has been approved and a Standard change may then simply be submitted for Post Implementation Review.

**Work Order Change Management Process**

The wo\_cycle implements a change process for Work Order ACMDs for a manufacturing process. This is used by the **WorkOrder** class of ACMD.



Initially the WorkOrder is *Raised* reporting a requirement to manufacture one or more Assemblies. The Assemblies to be manufactured should be specified as baselines affected. These may be meta or part release baselines which should be of class Assembly. These baselines should identifying the part versions which are to be manufactured.

Before the WorkOrder may be approved at least one Assembly baseline affected must be specified.

Once *Approved* the quantity of assemblies to be manufactured should be specified on the **Quantity** field prior to progressing to *UnderWork* to indicate that the items are being manufactured.

On entry to *UnderWork*, for each Assembly baseline affected, **Quantity** ProductionItem class instance baselines will be created containing new instances of the versions specified in the Assembly baselines. The ProductionItem class baselines are associated with the WorkOrder as baselines solved.

The ProductionItem baselines will be given the same basename as the corresponding Assembly baseline and will have a version of:

```
Instance.<number>
```

When the manufacturing process is complete the WorkOrder may be progressed to *Complete*.

## ITIL Baselines

### Baseline Classes for ITIL

The ITIL configuration defines the following classes of baseline.

#### Release

Defined for baselines which are to be processed using the release process implemented by the **ITIL\_release\_cycle**.

#### Assembly

Defined for baselines of hardware used in manufacturing to define the parts which make up an assembly. May be used for part or meta baselines. When used in conjunction with WorkOrder class ACMDs these may be used to automatically generate ProductionItem baselines of instances for the assembly.

#### ProductionItem

Defined for baselines of instances of hardware which are to be processed using the the **production\_item\_cycle**. When used in conjunction with WordOrder class ACMDs, baselines of this class are automatically created by the WorkOrder

### Baseline Fields for ITIL

Various arbitrary fields are defined for the **Release** class of baseline in the standard configuration - these may be added to or modified according to requirements.

The fields are used by the **ITIL\_release\_cycle** to provide information required to implement the release process. The fields are:

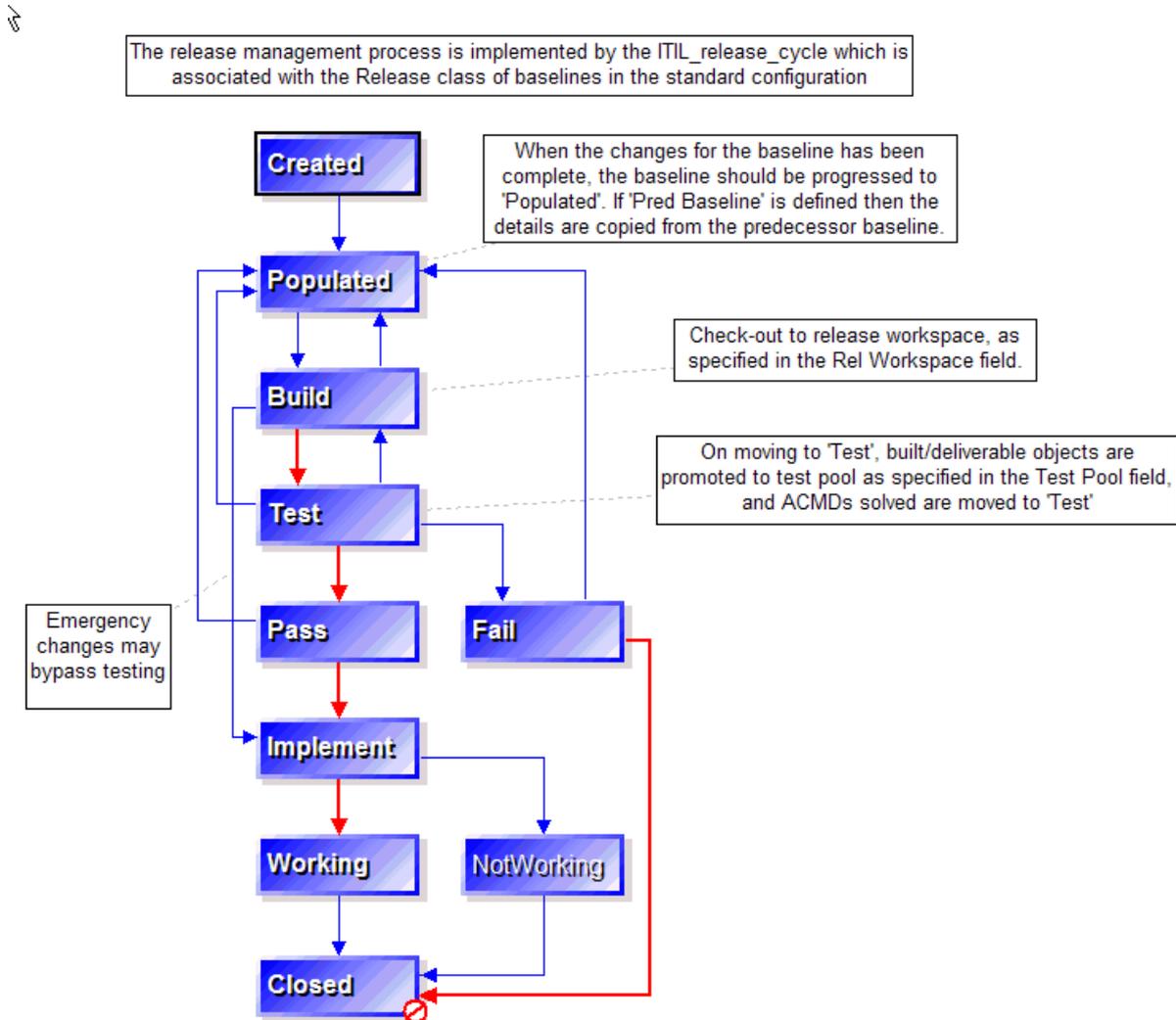
<b>Rel Dir:</b>	defines the release directory for the product final release
<b>Pred Baseline:</b>	defines the predecessor baseline. i.e. the baseline from which this one was created. A value of <b>(Not Applicable)</b> may be used if there is no previous release, otherwise it should contain the name of another release baseline
<b>Rel Workspace:</b>	defines the release workspace in which the product will be built for release purposes. The workspace name must start with <i>'rel'</i> .
<b>Test Pool</b>	defines the pool which is to be used for system testing the release. The pool name must start with <i>'test'</i> .
<b>Target Release:</b>	defines the release that RFCs to be included in this baseline will have as their Target Release.

### Baseline Release Management Process for ITIL

The release management process is implemented by the baseline life-cycles defined for each class of baseline.

#### Software Release Management Process

The release management process is implemented by the **ITIL\_release\_cycle** which is associated with the **Release** class of baselines in the standard configuration. The **ITIL\_release\_cycle** is defined as shown in the diagram below:



Initially the baseline is in a status of *Created* which indicates that the release is planned for some time in the future. The baseline should be created using **Add Baseline** with **Header Only**.

Work for the baseline is undertaken against RFCs which must have the baseline as their **Target Release**. RFCs are ready for release/inclusion in the baseline when they are in a status of **Build**. When the work for the baseline has been completed (i.e. all RFCs with the baseline as their Target Release are in a status of Build) the baseline should be progressed to *Populated*. If **Pred Bline** is defined then the details are copied from the predecessor baseline.

The RFCs which are to be released are then prompted for using the **ACMDs For Baseline** command: RFCs which are in a status with the **Release** attribute (Build) will be offered for release. The baseline

details are modified according to the CI's modified to implement the selected RFCs and each RFC is associated with the baseline as a *baselinessolved*. The baseline is also locked against further change.

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 workspace specified in the **Rel Workspace** field (if this is defined) ready for building; e.g. compile source. For efficiency **Rel Workspace** is actually updated to be contain the baseline - i.e. only parts not already checked out and at the correct version will be checked out. Any parts checked out which are not in the baseline are checked in. The **Rel Workspace** is therefore maintained with the baseline in it allowing for incremental releases and modifications to the current baseline without having to re-check out everything.

Once all build processes have been completed and the release is ready for test the status may be progressed to *Test*. If both the **Rel Workspace** and the **Test Pool** fields are defined then all deliverables are promoted from the **Rel Workspace** to the **Test Pool**.

Deliverables are defined as:

- All files in **Rel Workspace** which match one of the patterns defined in **Global\_Derived\_Deliverables\_Patterns** which is defined in the **Other** configuration options.
- Any files which correspond to parts which have the **Deliverable** field set to **Yes**.

Derived deliverables are moved whereas source deliverables are copied to the **Test Pool**.

All RFCs included in baseline (i.e. with this baseline as a baseline solved) have their status changed to *Test*.

If it is desired to delete all intermediate object files (\*.obj), then the code for this needs to be uncommented in the **ITIL\_release\_cycle**.

The *Test* status may not be exited unless all RFCs associated with the baseline have been tested (i.e. progressed from *Test* to *Pass* or *Fail*). When all the RFCs 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 progressed to *Implement*. On implementation all files will be copied from the **Test Pool** to **Rel Dir** in a flat directory structure. All RFCs included in the baseline which have passed testing are progressed to *Implement*.

The release may then be implemented and should then be flagged as *Working* if the implementation was successful, or *NotWorking* if the implementation was not successful.

The *Implement* status may not be exited unless all RFCs have been progressed from *Implement*. All RFCs must be *Working* in order for the baseline to be *Working*.

After a period of time the baseline may then *Closed*, for example when it is superceded by another baseline.

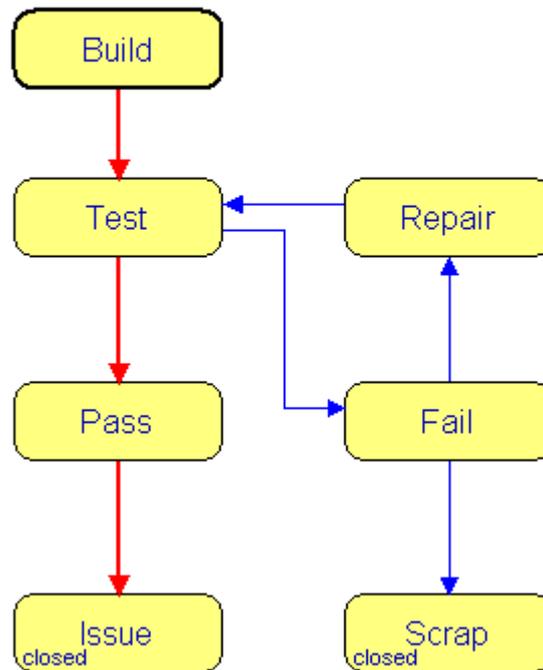
### Production Item Baseline Process

The life cycle for **ProductionItem** class of baselines is defined by the **production\_item\_cycle** in the standard configuration.

The life cycle is defined as shown in the [Production Item Cycle](#) diagram.

### Production Item Cycle

production\_item\_cycle for ProductionItem class of instance release baseline



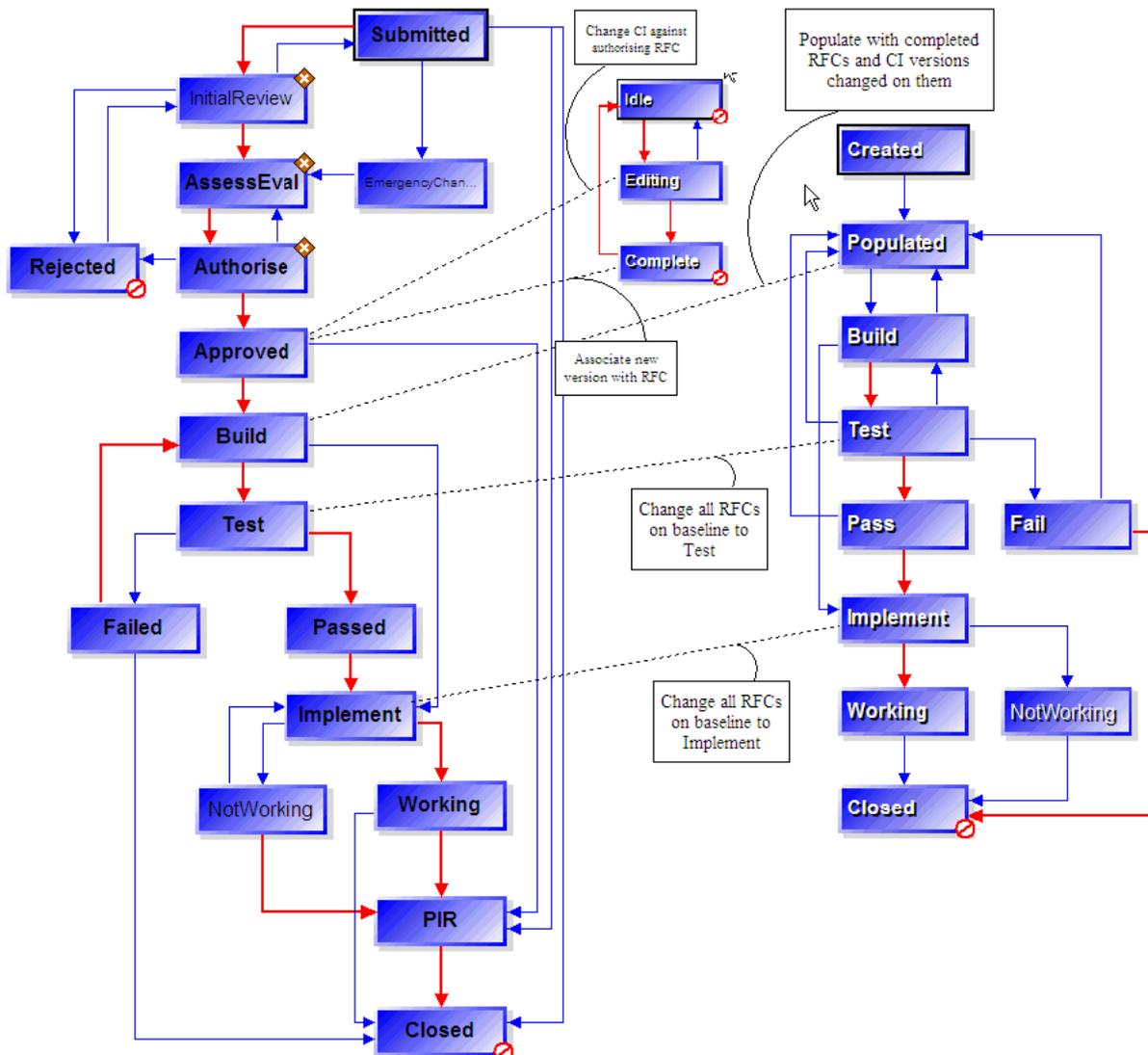
**ProductionItem** baselines may be created automatically by a **WorkOrder** class CR on entering status **UnderWork**. They will be automatically populated with instances created for each part in the **Assembly** class baseline associated with the **WorkOrder** as a baseline affected.

It will start in status *Build* to indicate that the object is being built or manufactured. When ready it is progressed to a status of *Test* to indicate it is being tested and undergoing quality control.

It then either *Passes* or *Fails* testing. If it passes it may then be *Issued*.

If it fails testing then it may either be *Repaired* or *Scrapped*.

## Summary of Cycle Relationships within ITIL for the Normal Change Process



### ITIL Roles

The ITIL configuration defines the following roles.

<i>Builder</i>	Members of the Build Team. Used by the Technical Team field on an RFC to nominate personnel responsible for effecting the change.
<i>Business Executive</i>	Members of the Business Executive. Used by the Business Executive Change Authority field on an RFC. Business Executive may be involved in the authorisation of changes where this level of authorisation is deemed necessary depending on the type, size and risk of a change.
<i>CAB_member</i>	Members of the Change Advisory Board. Used by the CAB field on an RFC. The CAB required for each RFC is allocated on the Change Authority tab of the RFC according to the level of authorisation required for the change. The CAB hence makes a part of the Change Authority and hence is responsible for authorisation of a change - see Change Authority below.

<i>Change Authority</i>	The Change Authority is not implemented as an IntaChange role but rather is allocated for each RFC from members of Business Executive, IT Management, CAB and ECAB authorities on the Change Authority tab of an RFC depending on the type, size and risk of the change. The Change Authority is responsible for authorising RFCs
<i>Change_Manager</i>	The Change Manager is responsible for the Initial RFC Review, Assignment of Assessors, Assignment of Priority, Assignment of Change Authority, Scheduling of changes, Post Implementation Review, tabling of RFCs for CAB meetings, Assignment of ECAB, Coordinate build, test and implementation of changes, analyse RFCs for trends etc, Closing RFCs.
<i>ConfigurationManager</i>	The Configuration Manager is responsible for managing the CIs, interfaces with Change, Problem and Release Management.
<i>Deployment</i>	Members of Deployment are responsible for implementing changes. Used by the Implementors field on an RFC.
<i>Developer</i>	Users who are developers. Used by the Technical Team field on an RFC to nominate personnel responsible for effecting the change if application development is required as a part of the change.
<i>ECAB_member</i>	Members of the Emergency Change Advisory Board. Used by the ECAB field on an RFC. The ECAB may be used for authorisation of emergency changes and should be allocated on the Change Authority tab of an RFC for Emergency RFCs.
<i>IT Management</i>	Members of IT Management. Used by the IT Management Change Authority field on an RFC. IT Management may be involved in the authorisation of changes where this level of authorisation is deemed necessary depending on the type size and risk of a change.
<i>Release_Manager</i>	Release Managers are responsible for releases of changes, interface with Change Management, Developers, Deployment and Testers
<i>Tester</i>	Members of the Test Team. Used in the Testers field on an RFC. Responsible for testing changes prior to implementation

### *ITIL Reports*

The ITIL configuration has the following reports:

<i>crFSC.acr</i>	<i>Forward Schedule of Change (FSC)</i>	This report shows the RFCs which are scheduled to be completed in the future, shown by date
<i>crxIAcceptRate.rep</i>	<i>Change Acceptance Rate</i>	This report shows the number of RFCs Approved and the number Rejected
<i>crxICAAssessed.rep</i>	<i>Number of RFCs Assessed by Change Authority</i>	This report shows the number of RFCs which have been Assessed by the Change

crxIRFCRaisedByType.rep *Number of RFCs Raised by Type*

Authority per  
Change Category  
This report shows  
the number of  
RFCs raised per  
Change Type

## [IntaChange Integrate Template](#)

### *About IntaChange Integrate Project Template*

**AllChange** comes with several ready to use out-of-the-box configurations which can be used as they stand or can be tailored to individual requirements.

The IntaChange Integrate project configuration is based on a simplified version of the ITIL template. Please see the help on the [ITIL template](#) for details.

The IntaChange Integrate template includes only two classes of CR - the RFC class - which has been modified to link in with the RFC class in IntaChange, and the WorkOrder class for use in manufacturing. The Incident and Problem classes are not present in the IntaChange Integrate template.

IntaChange has an equivalent template to integrate with AllChange, using the RFC class as well.

#### **ACMDs**

- [ACMD Classes](#)
- [ACMD Fields](#)
- [ACMD Change Management Process](#)

#### [Integrated Process Overview](#)

### *IntaChange Integrate CRs*

#### **ACMD Classes (IntaChange Integrate)**

In the IntaChange Integrate configuration the **AllChange** term CR has been redefined to the term ACMD (AllChange Change Management Document)

The IntaChange Integrate configuration defines the following class of ACMD.

#### **RFC**

**RFCs** are requests for change. They authorise actual component changes. **RFCs** are associated with the **ITIL\_RFC\_Cycle** and are numbered with an RFC prefix followed by a five digit auto incrementing number, e.g. RFC00001.

#### **WorkOrder**

Defined for ACMDs for manufacturing, to be used with baselines of class Assembly and ProductionItem. WorkOrders are associated with the wo\_cycle life-cycle. ACMDs of class WorkOrder are numbered with a WO prefix followed by a five digit auto incrementing number e.g. WO00001

#### **ACMD Fields (IntaChange Integrate)**

The fields for the IntaChange Integrate template for RFCs are the same as the [ITIL template](#), except that many of them are read only fields as the values are supplied by IntaChange and should not be changed. In addition there are the following differences:

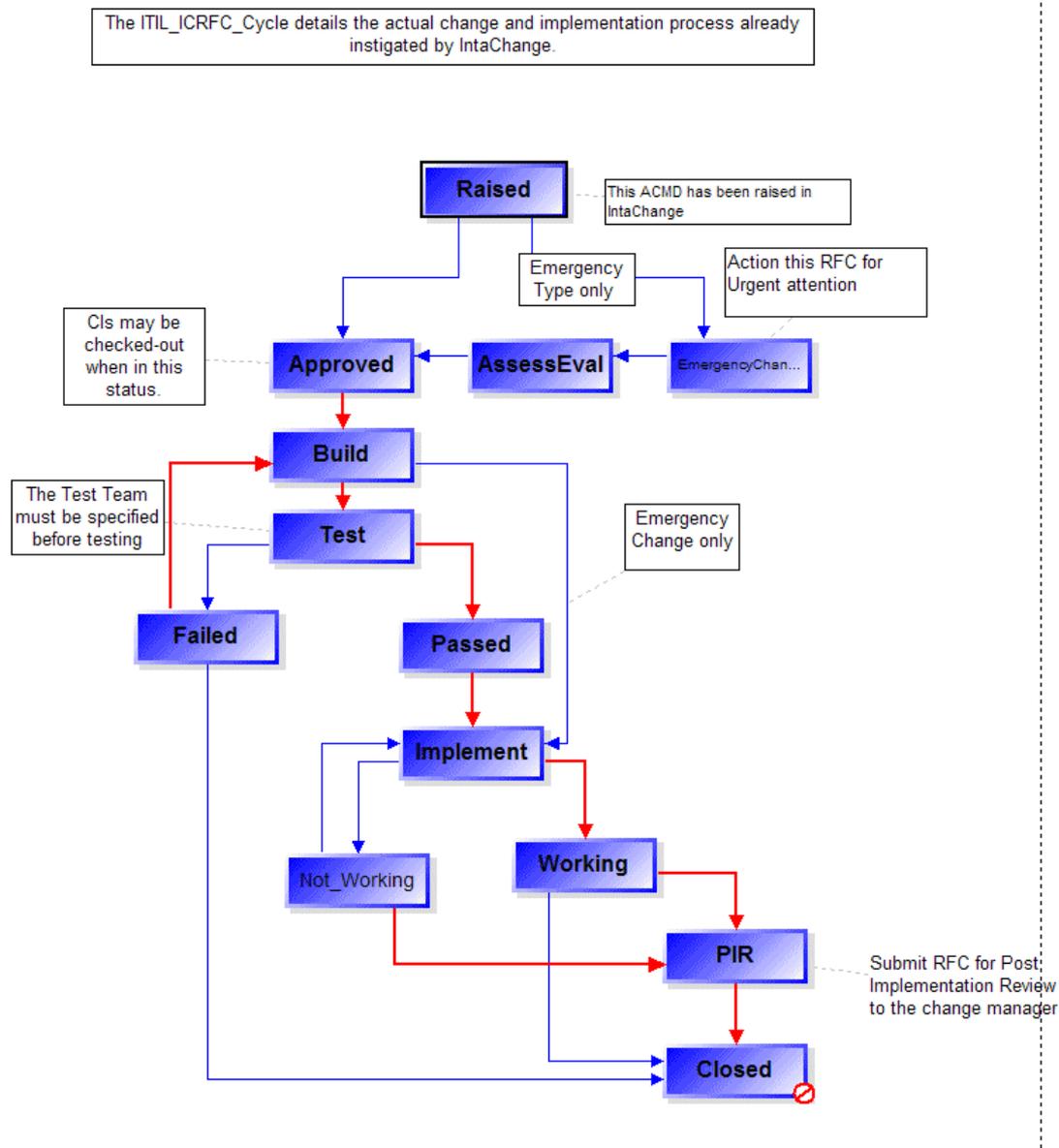
<b>Change Type</b>	Level of change - only <b>Normal</b> and <b>Emergency</b> change types will be reflected in <b>AllChange</b>
<b>Assessors</b>	Present in the ITIL template but not the IntaChange Integrate template
<b>Business Executive</b>	Present in the ITIL template but not the IntaChange Integrate template

<b>CAB</b>	Present in the ITIL template but not the IntaChange Integrate template
<b>ECAB</b>	Present in the ITIL template but not the IntaChange Integrate template
<b>IT Management</b>	Present in the ITIL template but not the IntaChange Integrate template
<b>Resource Costs</b>	Present in the ITIL template but not the IntaChange Integrate template
<b>Hardware Costs</b>	Present in the ITIL template but not the IntaChange Integrate template
<b>Software Costs</b>	Present in the ITIL template but not the IntaChange Integrate template
<b>Total Costs</b>	Present in the ITIL template but not the IntaChange Integrate template
<b>IntaChange Control</b>	Indicates that whether <b>AllChange</b> is in control of the RFC

The RFC Text tab has the same sections as the ITIL template.

### **ACMD Change Management Process (IntaChange Integrate)**

The change management process for the IntaChange Integrate template is a cut-down version of the RFC cycle in the ITIL template. Refer to the [ITIL template](#) for information about this cycle.



IntaChange creates a new RFC in **AllChange** when it reaches the **Approved** or **EmergencyChange** status in IntaChange. The **AllChange** status is then automatically progressed from **Raised** to the **Approved** or **EmergencyChange** status as appropriate to correspond to the IntaChange status. The RFC is synchronised back to IntaChange again on the **Implement** or the **AssessEval** status. *Note that prior to progressing the status from **EmergencyChange** in **AllChange** to **AssessEval** certain fields in the IntaChange RFC must have been completed to allow the IntaChange RFC to progress to the **AssessEval** status.*

For an Emergency Change on progressing from to the **Approved** status in IntaChange the corresponding RFC in AllChange is updated and control is passed back to AllChange.

At this point the IntaChange Control field is automatically set to indicate that control has gone back to IntaChange and no further progressions should be done in AllChange. Moving the ACMD to the **Working**, **Not Working**, **PIR** and **Closed** statuses in IntaChange will update the status in AllChange each time.

During the AllChange life-cycle the following fields are synchronised back to IntaChange each time they are updated:

- **CR number** (to the IntaChange field AllChange RFC)
- **Summary**
- **IntaChange Control**
- **Scheduled Completion Date**
- **Scheduled Implementation Date**
- **Estimated Effort**

# Parts/Files and Database Relationships

## About Parts/Files and Database Relationships

As the **AllChange** Administrator, it is important to understand the relationships between parts, files and the database.

This chapter discusses those relationships and the control that the administrator has over them.

### The Database

The database comprises 4 primary areas each of which is affected by underlying AllChange commands:

**Parts:** Holds information about the parts and meta data associated with them. The primary underlying commands which affect this table are: [add](#), [delete](#), [rename](#), [alter](#), [issue](#), [return](#), [newversion](#), [alterversion](#), [deleteversion](#)

**Baselines:** Holds information about baselines. The primary underlying commands which affect this table are [baseline](#), [alterbaseline](#), [deletebaseline](#), [renamebaseline](#)

**CRs:** Holds information about CRs. The primary underlying commands which affect this table are: [newcr](#), [altercr](#), [deletecr](#)

**Issues:** Holds information about all check-outs. The primary underlying commands which affect this table are: [issue](#), [return](#), [alterissue](#)

Note that the terms Part, Baseline, CR and Issue may be mapped to alternative terminology using [Nomenclature Mapping](#). The term *Issue* is mapped in the out-of-the-box configuration to *Check-out* and the documentation refers to *check-outs* rather than *issues*. However it is important for the administrator to understand that the underlying database table is the issues table.

The underlying commands which affect the database may be presented to the user as a higher level function (e.g Check In/Out instead of Issue/Return) but ultimately those higher level functions will invoke the underlying command to effect the operation and changes to the database.

### Parts, Files and the Database

For each part there is a corresponding directory or file which contains the physical contents of the part and, for components, the contents of the versions of the file.

- Subsystems → directory containing version history (VC) files for the components in the subsystem
- Components → version history (VC) files containing versions

This set of files and directories is the secure store (or archive) of the parts, and in many ways should be regarded as part of the database. However, the files are *not* physically held in the **AllChange** database.

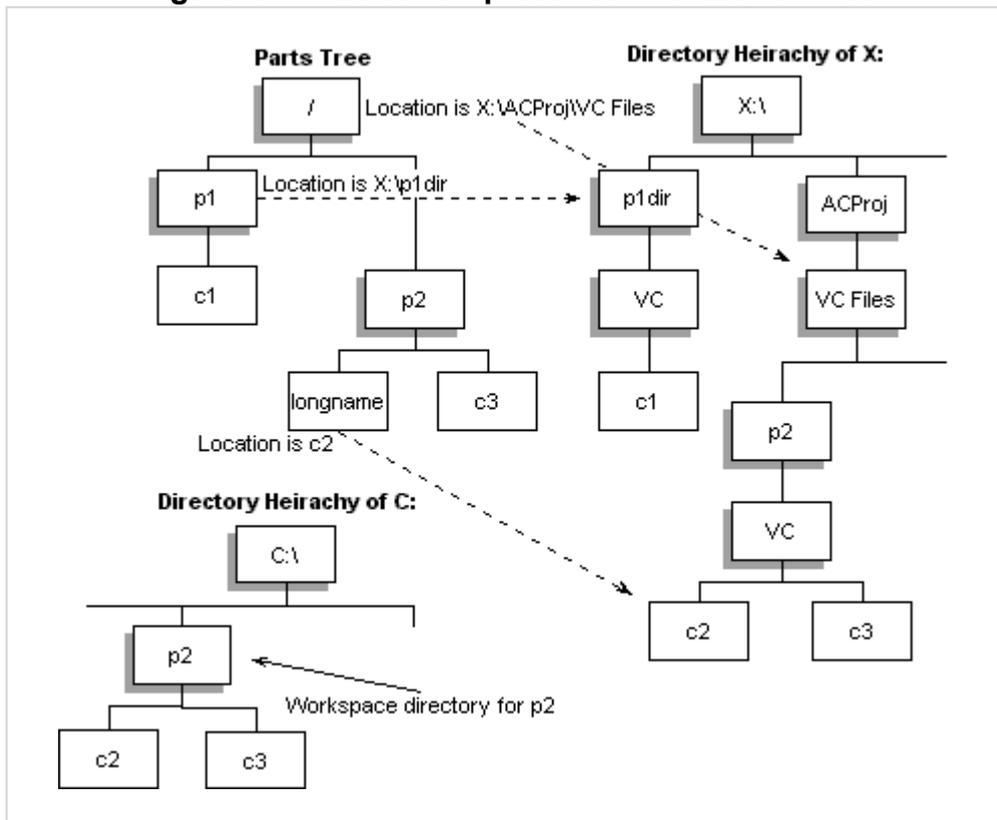
*The physical location of the secure store is determined by the **Location** field of the parts.*

The default is to set the location field of the root part (/) and allow all further locations to be defaulted and all version history files will be stored within the initially chosen directory hierarchy. The location set by creating a database is `$$ACPROJVCDIR\VCFILES`. This uses the [variable location](#) feature to mean that the actual location path starts with the value of the `ACPROJVCDIR` environment variable. ACE (and ACC) automatically create this environment variable and set it to the path to the AllChange project directory that they are currently using to access VC files, taking into account whether running VC client/server. This means that the location field should not need to be modified when changing between different client/server access modes or indeed when moving/copying the project directory into different directories. This is the recommended setting for locations.

In addition to the secure store for the parts, there will be workspaces where local copies of versions of the files reside.

In [Figure 5.1](#) possible relationships between a parts tree and the VC files and workfiles corresponding to it are shown.

**Figure 5.1: Relationship between Parts and Files**



In the normal case there is a one-to-one relationship between each part and each directory or file. However under certain circumstances it may be desirable to:

- have a greater depth of part hierarchy than directory hierarchy
- place some parts in a different location from others
- have part names which exceed the name length restrictions placed by the operating system for files

All of the above scenarios may be achieved using **AllChange** by setting the location field as required.

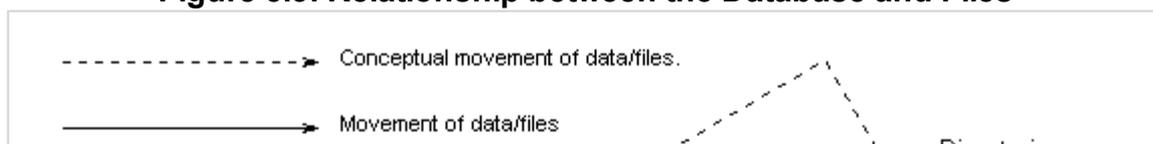
[Figure 5.2](#) shows the part names for the example system shown in the [Figure 5.1](#) together with their location field and illustrates the operating system path of each component's VC file. The location column is blank where the location field of the part is empty. The operating system is Windows, under which the distinction between **AllChange** part names and operating system path names should be clear.

**Figure 5.2: VC File locations**

Part	Location value	VC Dir/ File name
/	X:\ACProj\VCFiles	X:\ACProj\VCFiles
/p1	X:\p1dir	X:\p1dir
/p1/c1		X:\p1dir\VC\c1
/p2		X:\ACProj\VCFiles\p2
/p2/longname	c2	X:\ACProj\VCFiles\p2\VC\c2
/p2/c3		X:\ACProj\VCFiles\p2\VC\c3

[Figure 5.3](#) shows the relationship between the various aspects of the **AllChange** database and directories containing files. It also shows the movement of data between these areas.

**Figure 5.3: Relationship between the Database and Files**



## Variable Locations

Locations are used at various times in **AllChange** to tell the system where VC and workfiles actually reside.

The simplest file locations are fixed, e.g. the location field for the root part, /, might be /public/VCFILES or K:\PUBLIC\VCFILES. This means that operating system paths can never vary no matter where the database is being accessed from. This imposes restrictions when using the system from several clients on a network, e.g.:

1. under NFS, all machines have to use the same path to access the files; however, some networks prefer to utilise different paths from different machines;
2. on PC networks, the drive letter used to access the server directory has to be the same on all machines and at all times; however, some networks allow the letter to vary from machine to machine or from login to login.

**AllChange** allows (certain) path names to *commence* with a "variable" so that the actual operating system location is only fixed while an instance of **AllChange** is running. The path names which allow this feature are:

- the location field of part records
- the directory associated with a workspace (specified in the workspaces database)
- the directory associated with a pool (specified in the pools database)

If any of these pathnames commences with \$\$ (two dollar characters), the word following the \$ is taken to be a "variable" which is expanded to produce the actual path. Variables are, as usual, read from the environment or from the .ini file ([Common] section). So a variable path may look like \$\$PRODUCT or \$\$TOP/subdir.

Variable locations in part location fields are expanded when accessed; variable locations in workspace and pool locations are expanded when the database is read in. Variable locations follow the same inheritance rules as fixed locations, e.g. if the location of / is \$\$TOP then (unless overridden) the location of /subsys will be \$\$TOP/subsys (or \$\$TOP\subsys).

Fixed locations are easier to manage than variable ones so should be preferred if possible. If variable locations are used *it is very important that all users of the system have the correct resolution of all variables*. Although any part can have a variable location field and users can set variable location field on parts, it is envisaged that only the Administrator will use this facility and only on the root (or maybe other top-level) part. As standard the location of / when a database is created is set to \$\$ACPROJVCDIR\VCFILES where the ACPROJVCDIR variable is set by **AllChange** on start up to the location of the project directory. This allows the AllChange project directory to be moved without the need to change the location of the / part.

## Regular Administrative Tasks

### About Regular Administrative Tasks

Configuring the **AllChange** system is a major part of the **AllChange** Administrator function. However, once the major part of this has been achieved there are still some regular administrative tasks to be performed:

- Ensure that the database and associated files are backed up and the log files cleared out, see [Backing Up](#).
- Ensure that the database and external version history (VC) files are consistent. This is performed by running the `pachkvc` report on the parts database. Any discrepancies will be flagged and should be resolved, see below.
- Ensure that the **AllChange** configuration files themselves are maintained under version control. We recommend that **AllChange** itself is used for this purpose but with a very simply *out-of-the-box* project configuration.

### Resolving Discrepancies

It is possible for the database and the external version history files (VC Files) to get out of synchronisation with each other, i.e. The information about a part and the versions of that part do not agree with the actual versions stored in the VC File. It is a fairly rare occurrence and would usually only be caused by an unexpected termination of the client program when in the middle of a check in or check out operation.

The report `pachkvc` (available from the Format List dialog on the Report Part dialog) will detect any discrepancies and give information about the versions in the parts database and the versions in the VC File.

Any discrepancies reported should be resolved as soon as possible by making appropriate adjustments to the parts database and/or the VC File. Intasoft support will be happy to assist you in identifying the appropriate corrective action. The most common discrepancies and the appropriate corrective action are shown below.

### Common Discrepancies

Below are shown some common discrepancies which may occur, however, this is by no means a definitive list and if your discrepancy does not match exactly the situation described below or you are unsure, please contact Intasoft support for assistance with resolving your discrepancy.

#### Extra version in VC file than in AllChange

This can occur in two different cases:

1. The part is still checked out for edit in **AllChange** but has been putaway (stored) in the VC File. To resolve the discrepancy:
  - Switch off command actions (Misc | Set System Flag)
  - Return from edit (Part | Return) - note that this is only available to Admin users
  - Ensure the status of the version is correct
  - Switch actions back on
2. The part is not checked out for edit but the version has the NoVersion flag set. To resolve the discrepancy:
  - Change the flags on the version using Part | Version | Alter Flags selecting No Flags

#### Extra version in AllChange than in VC File

This occurs if a failure occurred during a check in before the workfile was stored in the VC file. To resolve the discrepancy:

- Switch off command actions (Misc | Set System Flag)
- Remove the **AllChange** Version using Part | Version | Delete Version

- Switch actions back on
- If there is a workfile that needs to be stored, check it in in the normal way

## Backing up

The **AllChange** system should obviously be backed up regularly and frequently since the information it contains will probably change a great deal. The System Administrator is responsible for defining what needs backing up, writing scripts to accomplish this and ensuring that backups take place at the right times.

There are quite a few groups of files which must be borne in mind when deciding what to back up, including:

- the actual SQL database itself
- the actions log file (`acacts.log` in the project directory).
- **AllChange** system/configuration files which have been tailored/added (in the project directory)
- VC version history files associated with parts in the database (usually in `VCFILES` subdirectory of the project directory)
- attachment files associated with CRs and baselines (in `crattach` and `blattach` subdirectories of the project directory)
- archive files containing archived CRs, parts and baselines
- workfiles in user workspaces
- objects promoted to shared pools
- objects copied to release directories

The Administrator is responsible for ensuring files are not in use at the time they are backed up. In particular it is very important that no users are accessing the **AllChange** database and log files at the time they are backed up so that a consistent set is captured: back up at night and create the database lockout file (see [Lockout Times](#)).

A sample script file, `BACKUPAC.BAT`, is supplied with the **AllChange** to show how the **AllChange** database files may be backed up and the log files cleared out. (This in turn uses `BACKUPDATABASE.BAT` which is supplied in the Database subdirectory of the **AllChange** system directory.) An Icon for this script is created on installation, but both the icon and the script will probably need modification before use.

The modifications required are:

- The path to the **AllChange** project directory to be backed up must appear on the command line.
- The SQL Server instance name must be supplied on the command line (as specified in the project definition)
- The SQL Database name must be supplied on the command line (as specified in the project definition)

As supplied it performs the following operations:

- Creates a lockout file to prevent users accessing **AllChange**
- Backs the database to the SQL server default backup location
- Copies the `acacts.log` file to the backup subdirectory of the **AllChange** project directory
- Empties out the log
- Restores any previous lockout file and allows users back into the database

The script may be used in combination with backing up the additional associated files as mentioned above. It is important that the in particular the following are backed up together to ensure a consistent data set:

- SQL Database
- `acacts.log`
- VC Files
- Attachment files

## Backing Up and Restoring the Database in ACCONFIG

In addition to the script there is a facility in ACCONFIG to Backup or Restore the SQL Database on an ad-hoc basis. The ACCONFIG **File | Update Projects | Database** tab has a **Backup/Restore...** button. This allows for very simple *ad-hoc* backups. It is mostly intended for testing/evaluation. In a production environment, backups should be maintained by your SQL Administrators, probably using SSMS or scripts, and scheduled on a regular and unattended basis.

If you do choose to click the **Backup/Restore...** button, it presents a simple "wizard" offering either backup or restore:

### Backup

- Choose a directory where the backup file(s) will be stored. If you select (**Default**), SQL Server will place the backup in its own default directory (e.g. **C:\Program Files\Microsoft SQL Server\MSSQL10\_50.ALLCHANGE\MSSQL\Backup**). Note that this file path is from the SQL Server machine's point of view, which may (well) not be the machine from which you are running ACCONFIG; furthermore, it is SQL Server which performs the backup on your behalf and it will use/need its own permissions to the directory/file, not yours. The backup file for the database will be named *database-name.bak*, e.g. **AC\_AllChange.bak**). If the SQL database's *Recovery Model* is set to **Full**, there will be a second file named *database-name.trn* for the transaction log.
- If you have previously backed up the database to the selected directory, you can choose either **Append** (new backup will be appended to previous ones already in the file, preserving them) or **Overwrite** (overwrites any previous backups, losing them). If the backup file does not exist (no previous backups) it does not matter which one you choose.

If all goes well a "successful" message will be presented. If there is a problem you will see details of the error, which you will need to inspect carefully to determine where the fault occurred.

### Restore

- Choose the directory where the previous backup file(s) were stored.
- For **Backup of**, you need to stipulate the name of the SQL database which was backed up. In the usual case this will be the same database as you are restoring to, and you may pick (**Self**) to indicate this. However, it is also possible to take a backup of one database and then restore it to a different database (e.g. for producing a copy).
- For **File Number**, you need to stipulate which backup within a multi-set backup SQL should restore from. If your backups have always been with **Overwrite** you will always select **1** (there is only one backup in the set), but if you have used **Append** it may be a higher number (e.g. the highest number available for the most recent backup appended into the file).

*When you restore you are (obviously) completely overwriting the existing database, so exercise extreme care.*

If all goes well a "successful" message will be presented. If there is a problem you will see details of the error, which you will need to inspect carefully to determine where the fault occurred.

# The AllChange Configuration Facilities

## About The AllChange Configuration Facilities

The various aspects of the **AllChange** system which may be tailored to your requirements are described in the following sections. Some of the configurable aspects of **AllChange** allow code to be written using ACCEL, the **AllChange** Command Evaluation Language, which is described in [ACCEL — AllChange Command Evaluation Language](#).

All these facilities may be modified using the ACCONFIG tool which is introduced in [The AllChange Configuration Editor — ACCONFIG](#).

## Defining Access Control

### About Defining Access Control

**AllChange** provides facilities to allow access to the various functions and commands to be restricted/ permitted as required for each **AllChange** project .

The access control rules may be specified using user roles and groups.

Implementing the required access control rule requires defining the following:

- The users permitted to use **AllChange** — [User Registration](#)
- User groups — [User Groups](#)
- The times when **AllChange** is available — [Lockout Times](#)
- The roles to be used and the users allocated to those roles — [Role Definitions](#)
- The roles/ users which are permitted to perform each action — [Command Access](#)

The access control definition facilities are available from the **Access** menu.

### User Registration

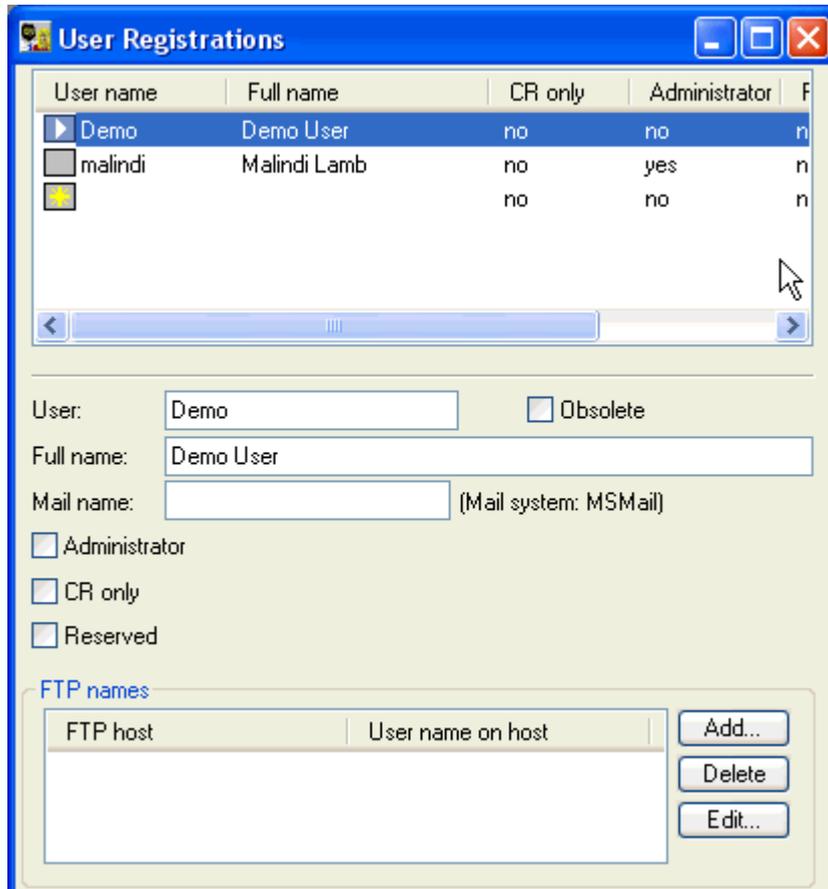
Full users and CR only users must be registered to the system as licensed users, see [Licensing](#) for details.

This requires that you register each user as an AllChange user in the User Registrations. Also if you are using the server logon facilities you will need to set up each user logon, see **Server Password**. If using the Web Browser interface to **AllChange** you also need to define the logon for this, see [AllChange Web-Browser Access](#).

Below are details of the user registration facilities.

The initial full installation gave you the opportunity to register yourself as an **AllChange** user. Further details about a user may be made and new users registered using ACCONFIG.

New users may be registered and the details of existing registered users modified using **Access | User Registration**.



Initially, if no users are licensed, you do not need to be a licensed user to use ACCONFIG. As soon as one or more users are licensed, then ACCONFIG requires you to be a licensed user.

To modify details of an existing user registration select the user in the list and then make the required modifications to the details shown in the bottom half of the window.

To add a new user select **Edit | New Item**, or select the blank item at the end of the list, then complete the details in the bottom half of the window.

For each user you *must* give the user name as **User**. User names must match **AllChange**'s idea of the username as described in [Operating System Specifics](#).

If a user who is not registered attempts to access **AllChange** and unlicensed access has not been permitted an error will be issued, otherwise read-only access will be permitted to unlicensed users.

If using the named user [licensing](#) scheme then when the maximum number of licensed users is reached, ACCONFIG will prevent you from registering any further users. Furthermore, ACCONFIG will prevent you from registering more fully licensed users than your licence permits.

For concurrent user licensing there is no limit to the number of users that may be registered, only in the number that may invoke the application at one time. The **Reserved** option allows specified concurrent licenses to be reserved for Admin access.

If you are using the Web Browser interface then in addition to registering users to **AllChange** you must register them as users of the Web Browser interface, see [AllChange Web Browser Access](#).

For each user the following additional information may be specified:

**Obsolete:**

If checked, marks this user as obsolete. i.e. this person is no longer an **AllChange** user. This allows the user to have their Full name specified, and displayed, without that user requiring a licence. In all other respects, these users are treated as un-licensed users, and will not appear in lists of users.

Also, obsolete users may not have an email address specified, and so will not receive email notifications. Any fields containing an obsolete user will be shown with the user name surrounded by ()'s, for example (Obsolete User)

**Full Name:**

May be used to specify the users full name, or name by which they are normally known. This is the name that will be used for display purposes wherever user names occur. If no Full Name is specified then the **User** will be used.

**Mail name:**

Should specify the users mail address for the mailing system that has been selected in the **Plan | Configuration Options**, see [Configuration Options](#). This will be used whenever **AllChange** sends a mail message to that user, for example, when a CR is assigned to the user.

**Administrator:**

Should be selected if the user is to be an **AllChange** administrator. This will affect the facilities available to the user in ACE and who has access to ACCONFIG. Only administrative users may access ACCONFIG.

As a special case, if no users are registered as administrative users, then all users may be administrators.

**CR Only:**

Should be selected if the user is to only have access to CRs for creation and modification purposes (CR Only users have read access to the other parts of the database).

**Reserved:**

For concurrent user licensing. Reserves a specified user license for Admin access, see [licensing](#).

**FTP Host/ User List:**

This information allows you to specify the users logon ID on other remote platforms. This only needs to be specified if the user is to make use of the FTP facilities within **AllChange**, see [Integrated FTP support](#). To add a new host/ user select the **Add**, to modify an existing entry select **Edit**.

Specify the name or IP address of the remote machine in **FTP host** and the users logon name in **Username**.

If a users logon ID is the same on the remote machine as on the workstation using **AllChange**, then it is not necessary to specify host/ username information.

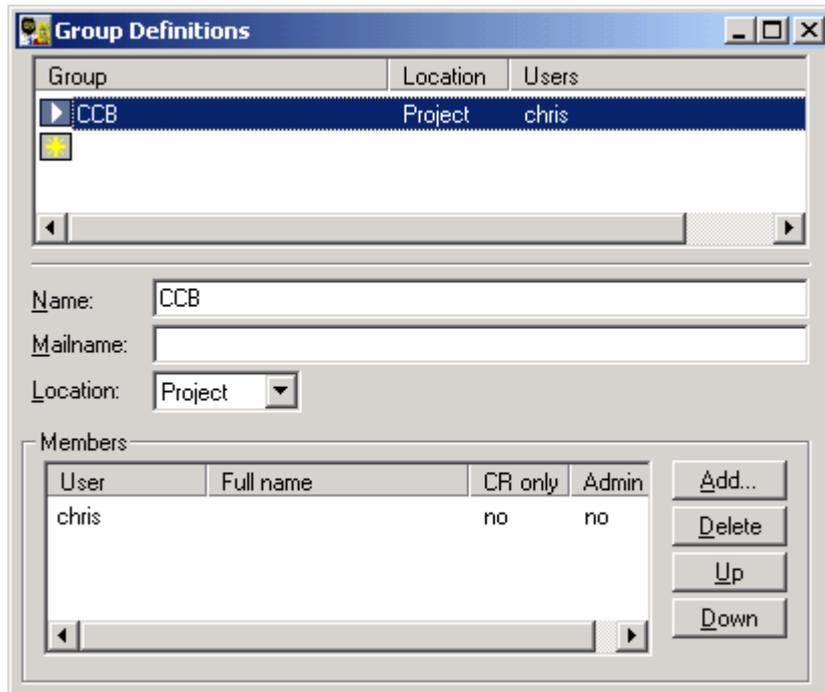
Use **Delete** to remove the currently selected host/ user entry.

The configuration files holding this information are `users.ac`, `userinfo.ac`, `mailmap.ac` and `ftpuser.ac` in the **AllChange** system directory .

**User Groups**

Groups of users may be defined for use in role definitions and other access control facilities.

To create or modify a group definition use **Access | User Groups**



Each group has the following information:

**Name**

This is the name of the group.

**Mailname**

This specifies the *group* mail name for the group, if there is one. If one is specified then when **All-Change** sends a mail message to the group, the group mailname will be used. If none is specified then each member of the group will be sent the message separately.

**Location**

This determines whether the group is system wide or just for this **AllChange** project. If a group is defined for both the system and the project then the project definition overrides the system definition.

**Members**

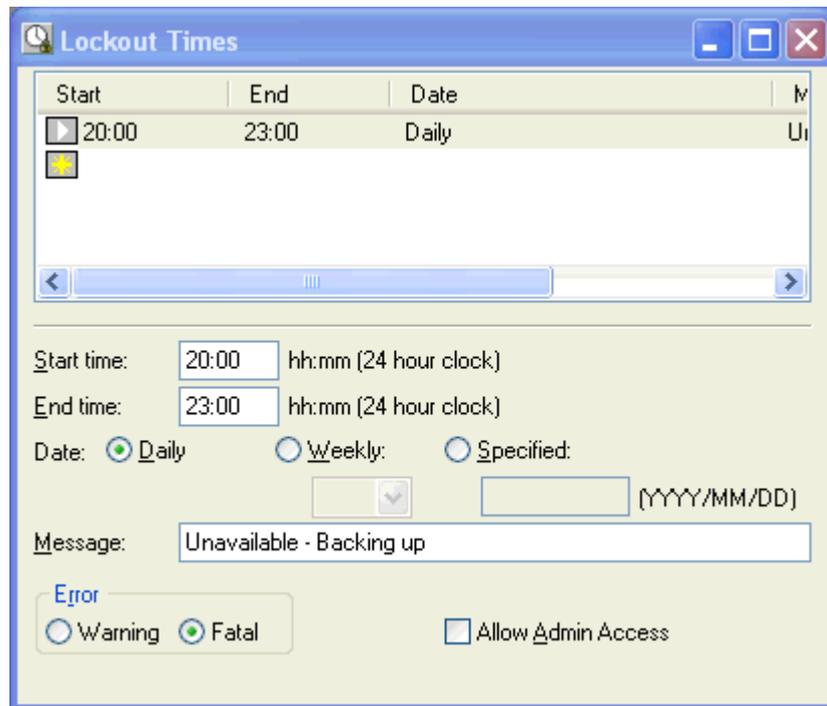
This is a list of the users who are members of the group. Members may be any of the *registeredAll-Change* users. Use the **Add** and **Delete** buttons to add new members to the group and remove members from the group. The order in which the members appears in the group may also be organised using the **Up** and **Down** buttons.

Where appropriate any groups defined will appear in lists of registered users.

The configuration file in which group definition are stored is `groups.ac` in the project director and system directory.

**Lockout Times**

The purpose of the lockout times facility is to prevent users from logging onto **AllChange** or to request/force users already logged on to log off during certain periods, e.g. in the middle of the night for backup purposes (see [Backing up](#)).



Any number of lockouts may be defined where each lockout should specify:

#### Start Time

#### End Time

These may be used to specify a daily time range during which **AllChange** is to be unavailable. The time should be specified in the format `hh:mm` in 24 hour clock notation. These are optional and if left blank imply *all the time*. These should be entered (and will be displayed) in ACCONFIG in local time and will be converted to UTC for internal storage.

#### Date

This may be specified as either **Daily**, **Weekly** or for a **Specified Date**. If **Weekly** is selected the day of the week may be selected. If **Specified** is selected then the date may be entered in the format `YYYY/MM/DD`.

#### Message

Specifies the message to be issued to the user.

#### Error

If **Fatal** is selected then when the lockout comes into effect a fatal error will be generated and the user is immediately logged off from **AllChange**. If **Warning** is selected then a warning is issued and the user is *not* logged off from **AllChange**.

#### Allow Admin Access

If selected then (on an entry with a fatal error) any user who is an admin user will receive only a warning message if the entry is activated, rather than being refused entry (or being logged off if already online) to AllChange, as happens to normal users. This is intended to be used to allow admin-only access while maintenance tasks are carried out; do not use this, for example, during backups when no access at all can be permitted.

A typical use of the lockout facility is if the System Administrator wishes to make the system temporarily unavailable to normal users during backup, maintenance etc. To disable user access a lockout entry should be created containing an explanatory **Message** for the user (e.g. `System down; retry after 9:00am`); no start or end time should be specified. (Don't forget to **Save** your changes. This results in the following: from now on if a user tries to invoke ACCor ACEit will issue a "Database not currently available" error message together with the lockout message, and then exit instead of opening the database. To re-enable remove the lockout entry (use the `DEL` key or **Edit | Delete Item**).

Note that changes to the lockout entries (when saved) will take immediate effect for new invocations of ACC and ACE but it has no effect on programs already running.

It is also possible to specify daily time ranges during which users are prevented from logging onto **All-Change** or requested or forced to log off from **AllChange**.

**Start Time** and **End Time** (which should be of the form HH:MM) specify a time range during which a lockout message applies. Whenever a user tries to log on the lockout entries are scanned (from top to bottom) to see whether the current time lies within any of these ranges. If it does, the corresponding message is issued and the user is not allowed on to the system (regardless of whether the entry constitutes a warning or a fatal error).

Periodic checking of the time ranges in the lockout file for users who are already on-line may be enabled by selecting **Check lockout times** in the [configuration options](#). While a user is on-line the system will then issue a warning message requesting the user to log off if the time lies within a warning range, and will actually automatically log the user off if the time lies within a fatal error range. Each message is issued only once.

So, suppose database backups are scheduled daily at 3am and take a few minutes to complete. You might create the following lockout entries:

Error	Start Time	End Time	Message
Warning	02:30	02:55	Message Backup due at 3am
Fatal	02:55	03:15	System currently being backed up

Users would not be able to log on between 02:30 and 03:15, receiving the appropriate explanation. Users already on-line would receive a warning message between 02:30 and 02:55 asking them to log off; if anybody were still on-line after 02:55 they would be immediately forcibly logged off with the appropriate message.

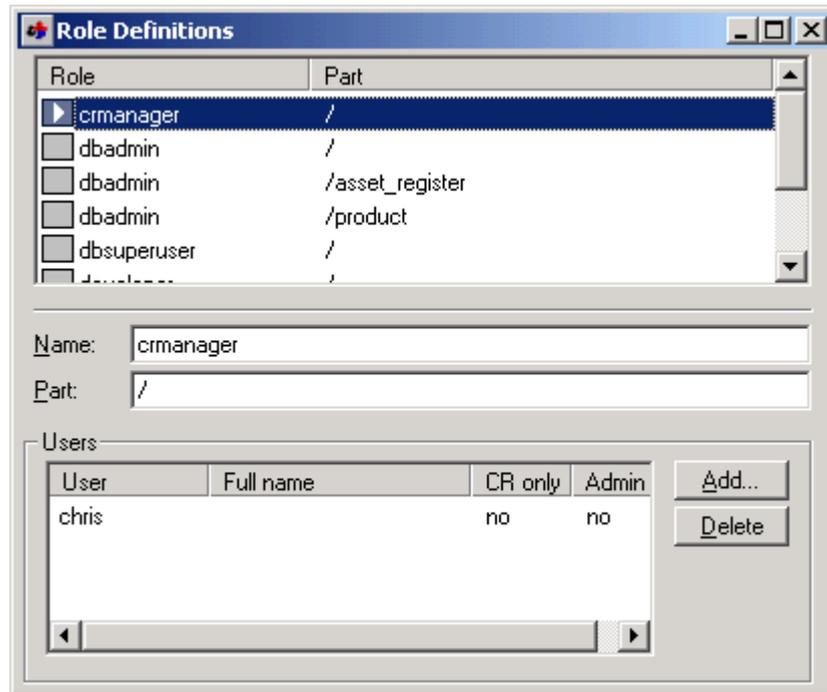
A few points. First, note that the lockout definitions are only read when entering ACE, so changing the lockout definition has no effect for users already on-line. The mechanism is suitable for regular backup scheduling which changes infrequently, but not for suddenly deciding to get current on-line users off the system. Second, forcibly logging off users is undesirable as the system cannot determine whether they are actually doing anything — it is preferable to get them to log off cleanly. If you choose to use "log off" (fatal) entries you must accept this: some sites may consider the security of knowing that a backup has taken place with no users on-line more worthwhile. The mechanism does work in the case where a user has gone home without logging off. Finally, ensure all users' clocks are correct — this applies generally to **AllChange** anyway.

If your **AllChange** users span multiple time zones then you should consider this when setting lockout times. Lockout times are *server* times and are stored in UTC in order to accommodate multiple time zones. For details of how dates/times are stored and interpreted see [Database Fields](#).

### Role Definitions

Roles are used to implement access restrictions within **AllChange**. Roles are defined to **AllChange** together with users permitted to have that role and a part associated with the assignment of that role. Roles effectively define groups of users with permission to perform various actions on certain areas of the parts database, see the Access Control and User Roles section of the **AllChange** User Manual.

New roles may be defined, users may be assigned to roles and existing roles may be modified by using **Access | Role Definitions**



Each role has the following information:

#### Name

An arbitrary name for a role (e.g. `developer`, `documenter`). The roles `dbadmin` and `dbsuperuser` have special significance. A role is only uniquely defined by its **Name** and the **Part** to which it is applicable, since there may be several roles of the same name for different parts. A role name must be comprised of alphanumeric characters and the `_` character and may not include a space character.

#### Part

The name of a part in the parts database for which a role is to be assigned. This may **not** be a uses type part. The part should be specified as the full partname e.g. `/product/sw/source`.

#### Users

A list of user/ group names which are granted the specified role for the specified part. If a group is specified then all members of the group are granted the role, see [User Groups](#) for details of group definitions. A user inherits a given role from a part downward in the parts tree until the *same* role appears again against a descendent of the original part specified. The same user may have a variety of roles over different regions of the parts tree. To add users use the **Add** button; this will present a list of registered **AllChange** users and defined groups. To remove users select the user and use the **Delete** button.

The *out-of-the-box* configuration includes definitions for the following roles for the whole part tree.

- developer
- crmanager
- release\_manager
- dbadmin
- dbsuperuser

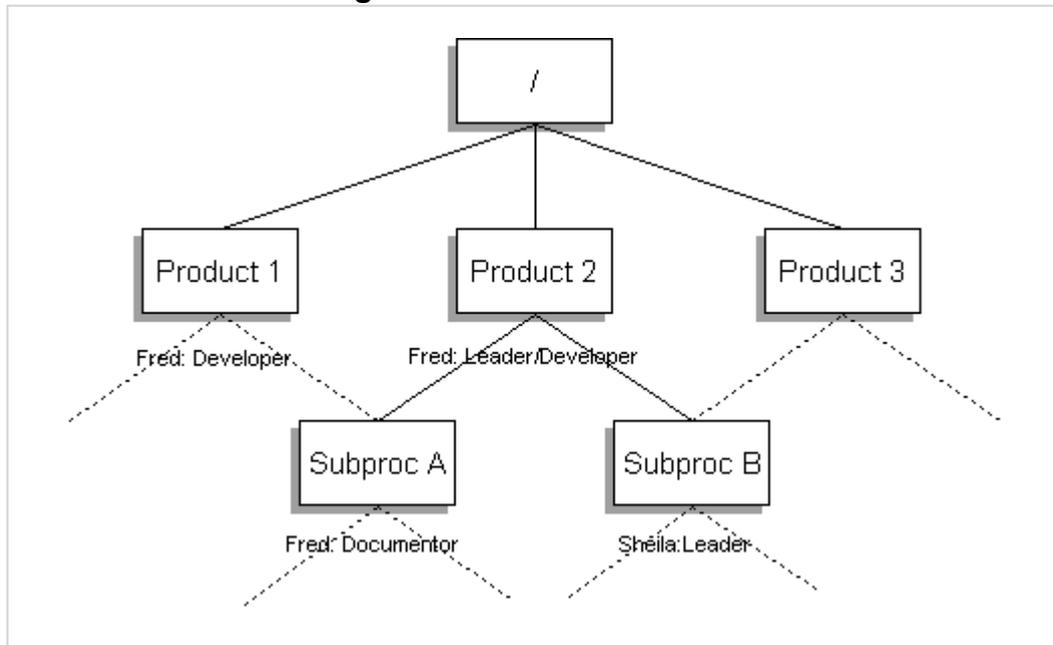
These are used in the *out-of-the-box* access control definitions. In order to use **AllChange**, users *must*, as a minimum configuration, be allocated to these roles.

A user inherits a role which has been assigned to him starting from the specified part downward in the parts tree until the same role is reassigned to other users for a descendent of the original part specified.

Note that this means that an **AllChange** user does not simply have a fixed role when using the system: rather he has (or does not have) a certain role *for a particular part*. For example, it is quite possible for a user to be a developer in one subsystem, a documenter in another, and have nothing to do with a third. It is possible to grant a user a particular role over all parts by assigning him the role for the root part (/) and then not reassigning the role elsewhere in the parts tree.

In fact, a user may have more than one role for a particular part. A user may, for instance, be a developer, documenter and project leader for a subsystem. The user will be granted all of these roles simultaneously for the part.

**Figure 8.1: Role Inheritance**



[Figure 8.1](#) illustrates the role inheritance mechanism over an area of the parts tree. fred has the role developer for the part /Product1 and all its descendants (indicated by the dotted lines). fred has the roles leader and developer for /Product2 and its descendants. From /Product2/subprodA downward fred gains the additional role documenter (retaining the previous roles). From /Product2/subprodB downward, however, fred loses the leader role (to sheila), but retains the developer role. fred has no roles in the /Product3 subtree.

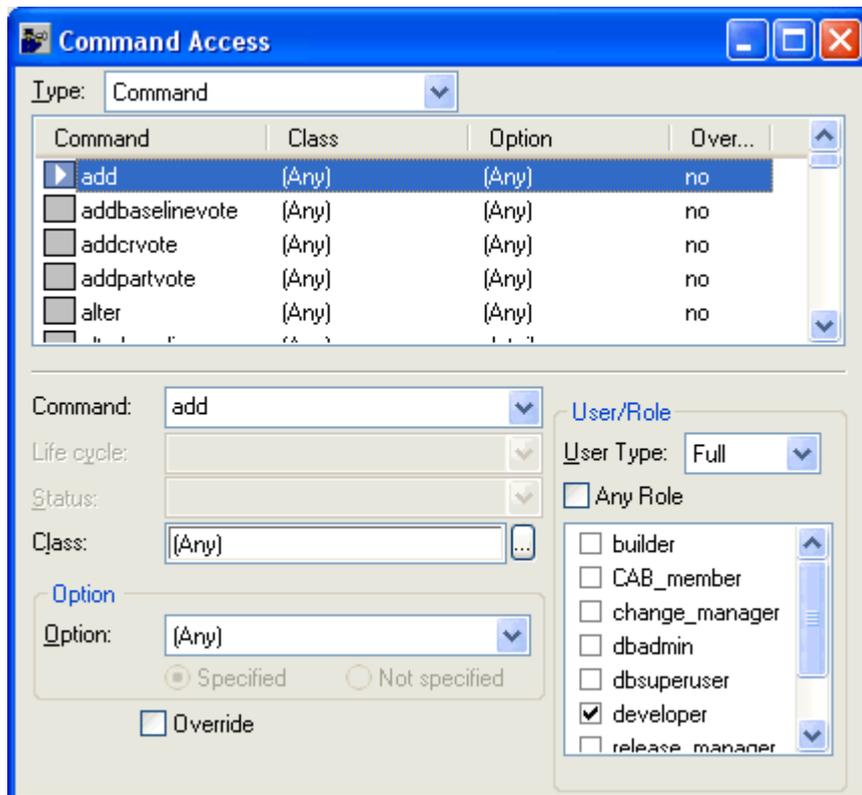
Note that defining a role and assigning users to a role does not in itself implement any access control based on that role. The role should be made use of in **Command Access** assignments. They may also be used in ACCEL code by use of the ACCEL `has_perm` function.

The configuration file in which the role definitions are stored is `roles.ac` in the project directory or system directory.

## Command Access

### About Command Access

The Command Access dialog is used to implement access restrictions for each **AllChange** command defining the users and roles which have permission to perform that operation.



Access control rules may be defined for different types of operation within **AllChange** as follows:

- Command:** underlying **AllChange** commands and pseudo commands
- Status change:** entry and exit to/from life-cycle statuses and ignore vote results for statuses for CRs, baselines and parts (components and subsystems)
- Function:** functions implemented as ACCELfunctions

For each of the above **Types** of operation the **User/Role** which may perform that operation may be selected.

The **User Type** should select the type of user who has access to the command.

If the **Any Role** check box is selected then *all* users of the specified type may perform this command (any role/ user selections are ignored).

If **Any Role** is not selected then a list of *special users* and defined roles is shown which should be selected as required to restrict access to that operation to those users/roles where the user has the specified **User Type**. If no users/roles are selected then *no-one* may perform that operation.

Where a role is required the role must be valid for the user for a *part* related to the action to be performed. For part based operations the part used will be the part on which the action is performed, for CR and baseline operations the part used will be the *toppart* associated with the CR or baseline.

The *special users* include:

**cr\_originator**

This may be used for commands which affect CRs to indicate that whoever was the originator of the CR has permission to perform the operation.

**cr\_assignee**

This may be used for commands which affect CRs to indicate that the current assignee of the CR has permission to perform the operation.

**bl\_user**

This may be used for commands which affect baselines to indicate that whoever created the baseline has permission to perform the operation.

**iu\_user**

This may be used for commands which affect the issues/check-outs database (i.e. **return**, **alter-issue**) to indicate that the user who checked out the part has permission to perform the operation.

**mo\_user**

This may be used for commands which affect monitors to indicate that the user who placed the monitor has permission to perform the operation.

**vd\_voters**

This may be used for commands which add votes to the votes database (**addbaselinevote**, **addcrvote**, **addpartvote**) to indicate the voters specified on the vote definition.

blvt\_user, crvt\_user, pavt\_user

This may be used for commands which affect votes which have been cast in the votes database ( **alterbaselinevote**, **altercrvote**, **alterpartvote**, **deletebaselinevote**, **deletecrvote**, **deletepartvote**, **readcrvote**, **readbaselinevote**, **readpartvote**) to indicate the user who cast the vote.

**ws\_users**

This may be used for the **attachws** command to indicate that users associated with the workspace have permission to attach to the workspace.

If more than one entry is applicable to a given operation, then **all** matching entries must succeed in order for the operation to proceed.

The **Override** flag may be used to modify this behaviour such that any entry with this flag set which passes its conditions then causes the operation to proceed overriding all later entries.

Additional information/criteria may be specified for each different **Type** of operation as detailed in the sections below.

The *out-of-the-box* configuration makes use of the developer, crmanager and release\_manager roles for the default access control rules.

In outline the permissions are:

Role	Permissions
crmanager	assign/re-assign and modify CRs not currently assigned to them, delete CRs,
developer	create and modify parts, assign parts to baselines, check parts in and out
release_manager	modify baselines, delete baselines, use the release command , <i>build</i> release baselines (enter <b>build</b> status for baseline of <b>releaseclass</b> ), close baselines pass and fail release baselines, populate release baselines, make release baselines ready for test
User Type	Permissions
CR Only	may only read/view CR information
Full	may read/view all data

The command access definitions are stored in the configuration file `rolemap.ac` in the project directory or system directory.

**Command**

**Command** type entries may be specified for every underlying **AllChange** command line command, these are the commands which are executed by the dialogs/windows in the ACE user interface and may be invoked via a command line in the ACC command line interface. The command names are largely self explanatory using the following convention:

- command names without cr or baseline as a part of the name are commands acting on parts (e.g add)

- commands delete/alter exist for part (for which the command name is just add/delete/alter), cr (e.g. altercr) or baseline (e.g. alterbaseline)
- the command to add new items are `add` for adding a part, `newcr` for creating a new cr and `baseline` for creating a new baseline
- other command names are specialist to their function and largely self explanatory

Note that with the use of [nomenclature mapping](#) commands may appear with different naming to the underlying **AllChange** command name in ACE; for example the out-of-the-box configuration maps issue to check-out, thus the **Alter Issue** menu item will show as **Alter Check-out**. The underlying **AllChange** command, however, remains `alterissue` and this is what must be used in Command Access.

In addition there are a number of *pseudo* commands which exist to allow access control to be defined for special cases, examples include:

- `ignoreprotecttext` this allows the roles which may override the CR protected text facility to be specified - default is `dbsuperuser`. If more precise control is required, for instance to protect text when a CR is in a particular status, a condition may be added to the `ignoreprotecttext` entrycond in the [Command Definitions](#).

The condition should error in those conditions where the text is to be protected, i.e. `cr_status == 'InWork'` and `cr_assignee == 'fred'` etc. (Note that the error message will not be seen by users.)

The `ignoreprotecttext` entrycond is only entered if the current CR's class has the "Protect Text" attribute, and the 'ignoreprotecttext' rolemap entry allows text to be edited. i.e. this allows more protection to be applied, rather than revoking any protection set in the Command Access settings. In the entrycond, the current CR is the CR whose text protection is being checked.

An example entrycond entry to disallow text from being edited directly unless assigned to the current user, and the current status is either Submitted or Accepted (i.e. work hasn't started on it yet), is below:

```
if not is_in_group_list(user, cr_assignee) and
    not is_in_list(cr_status, "Submitted Accepted") then
    error("");
endif;
```

- `readbaseline`, `readbaselinevote`, `readcr`, `readcrvote`, `readinstance`, `readmonitor`, `readpart`, `readpartvote`, `readbaselinestatuslog`, `readcrstatuslog`, `readpartstatuslog`, `readbaselinevote`, `readcrvote`, `readpartvote`: these allow access controls to be specified for read permissions to parts, baselines, CRs, monitors, votes and status log entries. With these pseudo commands access may be restricted to the entire database (baseline, cr, part etc), to specific classes of that item, or to individual fields of that item. This allows different users to have different views of the data.

If an item/field may not be read/viewed then it will not appear in the user interface (ace). If fields are hidden, then their columns in browsers will show a blank entry, their controls will not be shown on viewers, their values will not appear in reports, and direct field references (e.g. `cr_summary`) will return empty. If all fields of an item are hidden then the whole item is excluded from browser lists, reports, and ACCEL 'each' loops. If items are excluded from browser lists due to read permissions, then the user will see a notification at the head of the list to inform them of this, but this behaviour may be modified with the `showitemshiddenmessage` pseudo-command described below. If no items of a particular type are to be shown, then the relevant list is completely hidden, or the viewer tab is hidden or disabled. As an example, if a command-access setting is added to restrict the

baseline filesaffected field to full users only, then a CR-only user will never see the **Files** tab of the baseline viewer. If however, it is restricted to the release\_manager role, for all user-types, the tab is added to the baseline viewer, but is enabled only when the user has the release\_manager role. If a viewer is opened on a hidden item then the usual viewer tabs are replaced with a message informing the user that they do not have permissions to view the item.

*Note that in order for these read permissions to come into effect the [Allow read permissions](#) configuration option must be selected.*

For part permissions see also Edit Part Read Permissions in the reference manual.

If more precise control is required, a condition may be added to the pseudo command entrycond in the [Command Definitions](#). The entrycond is called once per record and then for every field to determine its visibility. To test whether the entrycond is called for a specific field use `getarg(<field-name>)` which will return true if invoked for that field. Care should be taken when adding or modifying code in these conditions, as any field references will themselves be subject to these conditions, and this code will be re-entered. Fields not available for command-access in `acconfig` are not tested here.

To prevent read-access to an item, or a field, the condition should error. Note though that the text of the error will not normally be seen by the user, except when viewing an item for which any access is denied, where the error text is displayed on the viewer instead of the usual tab-pages.

**Example:**

To hide the text of CRs in the status of released, use the following code in the 'entrycond' section of the 'readcr' pseudo-command (note that the first three lines prevent the condition from being re-entered when referencing `cr_status` in the condition):

```
if getarg('status') then
  return ''; # stop the reference to cr_status evaluating this condition
endif;
if cr_status == 'released' and (getarg('text') or getarg('text_raw')) then
  error_map("You may not view the text of released #Cr_s#");
endif;
```

There may be time where code is executed in a secure context but the fields' read permissions should be honoured. To achieve this, there is an ACCEL function [apply\\_read\\_perms](#).

An ACCEL function [is\\_field\\_readable](#) allows code to determine whether a field or item is readable, by the current user, with their current role.

An ACCEL variable [read\\_perms\\_denied\\_message](#) returns the reason for denial for the last (unsuccessful) call to `is_field_readable`. If the call to `is_field_readable` returned 'true' then this variable returns an empty string.

- `partpermadd`, `partpermdelete`, `partpermedit`: these pseudo-commands allow the buttons on the Edit Part Read Permissions dialog to be restricted. Buttons on the dialog are enabled/disabled according to the user's role, and the part selected in the permission list. This allows users to be able to edit the permissions on parts in the part tree for which they are responsible, while not being able to edit other permissions.
- `showitemshiddenmessage`: this pseudo-command allows the message informing users that items have been excluded from browsers to be hidden, so that users are not aware that items have been excluded.

These entries also correspond to the entries in the [command definitions](#) file (`commands.ac`)

**Command** type command entries may specify a **Class** to which the access control restrictions apply.

Furthermore, the entries may be further refined by defining the command options to which a particular rule applies.

As an example, the supplied configuration restricts modification of version flags to `dbadmin` role (Command **alterversion**, Class **any**, Options **flags**). Whilst other **alterversion** operations are permitted by the `developer` role (specified as a second entry for the **alterversion** command). This may also be used to restrict read/view permissions on specific fields.

This may be supplemented by defining whether the roles specified define the requirements for permission to specify the option or omit the option.

For example it may be desirable to state that only a CR Manager may create a new CR without allocating a Priority. This may be achieved by selecting an **Option** of Priority and **Not specified** and a role of `crmanager`.

Alternatively you may wish to say that only a project manager may modify the priority of a CR. This may be achieved by selecting an **Option** of Priority and **Specified** and a role of project manager.

A special entry which does not correspond to an underlying **AllChange** command is called **ignore-protecttext**, and should be used to specify the permissions required to edit the text for a CR of class with the [Protect Text](#) attribute.

## Status Change

Four different status change **Types** may be specified for CRs, Baselines, Components and Subsystems. Additional information for these type of access control rules must be specified as follows:

### Action

This should specify whether the rule is to apply as an *entry* or *exit* condition to the status (i.e. permission to enter or exit the status) or *ignorevote* for the status (i.e. permission to ignore the results of a vote in the status).

### Life-cycle

This should specify which life-cycle the rule applies to

### Status

This should specify which status within the above **Life-cycle** the rule applies to

### Function

**Function** type access control rules may only specify the base access control information.

Note that if an ACCEL function invokes an underlying **AllChange** command, then a user will need permission to perform both the function and the underlying command(s).

## Defining your Configuration Management Plan

### About Defining your Configuration Management Plan

Your configuration management plan defines the processes you are going to use to implement your configuration management system.

You may define your *plan* to **AllChange** by defining your processes and specifying various options.

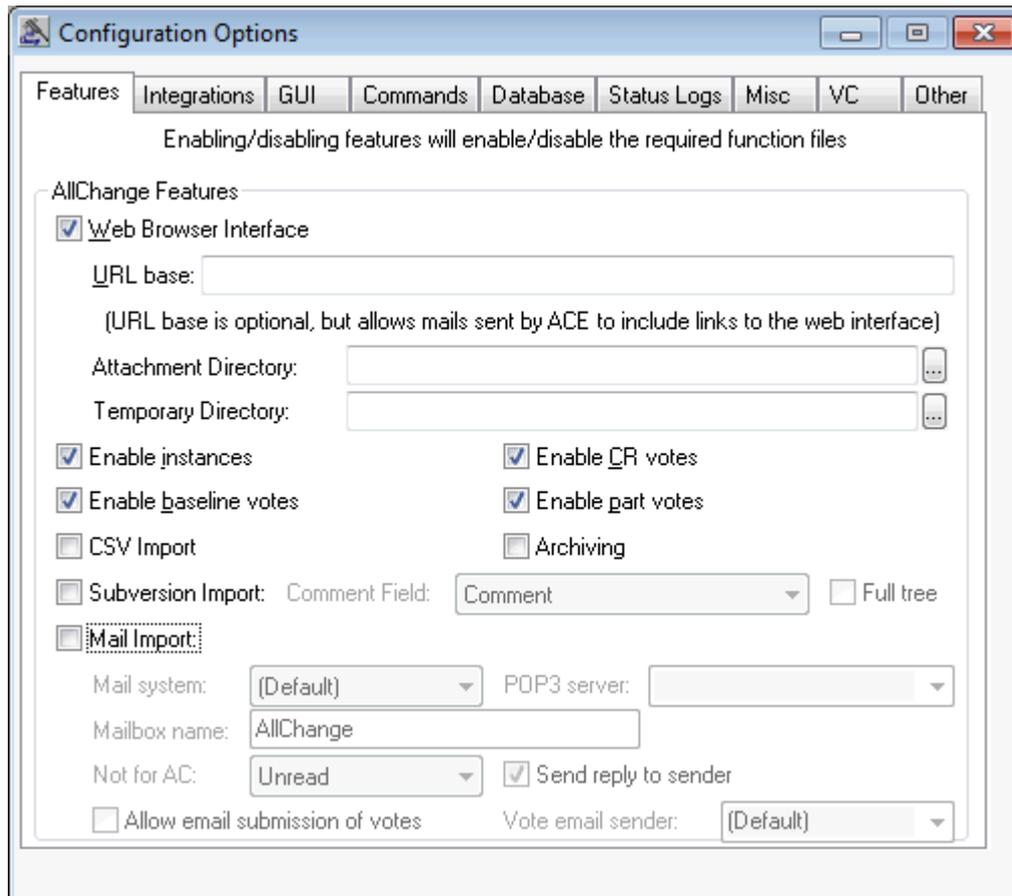
These facilities are available from ACCONFIG via the **Plan** menu.

## Configuration Options

### About Configuration Options

Various global **AllChange** configuration options may be specified which affect the behaviour of **AllChange**. These are available from **Plan | Configuration Options**.

The configuration file in which these settings are stored is `config.ac` in the project directory or system directory .



The configuration options available are presented on a series of tabs according to what aspects of **AllChange** they affect. The tabs are:

**Features**

Shows configuration options enable/disable certain **AllChange** features.

**Integrations**

Shows configuration options enable/disable **AllChange** integrations.

**GUI**

Shows configuration options that affect the ACEuser interface.

**Commands**

Shows configuration options that affect the behaviour of **AllChange** commands.

**Database**

Shows configuration options affecting the information stored in the database.

**Status Logs**

Shows the configuration options which enable/ disable the status logging facilities.

**Misc**

Shows various miscellaneous configuration options.

**VC**

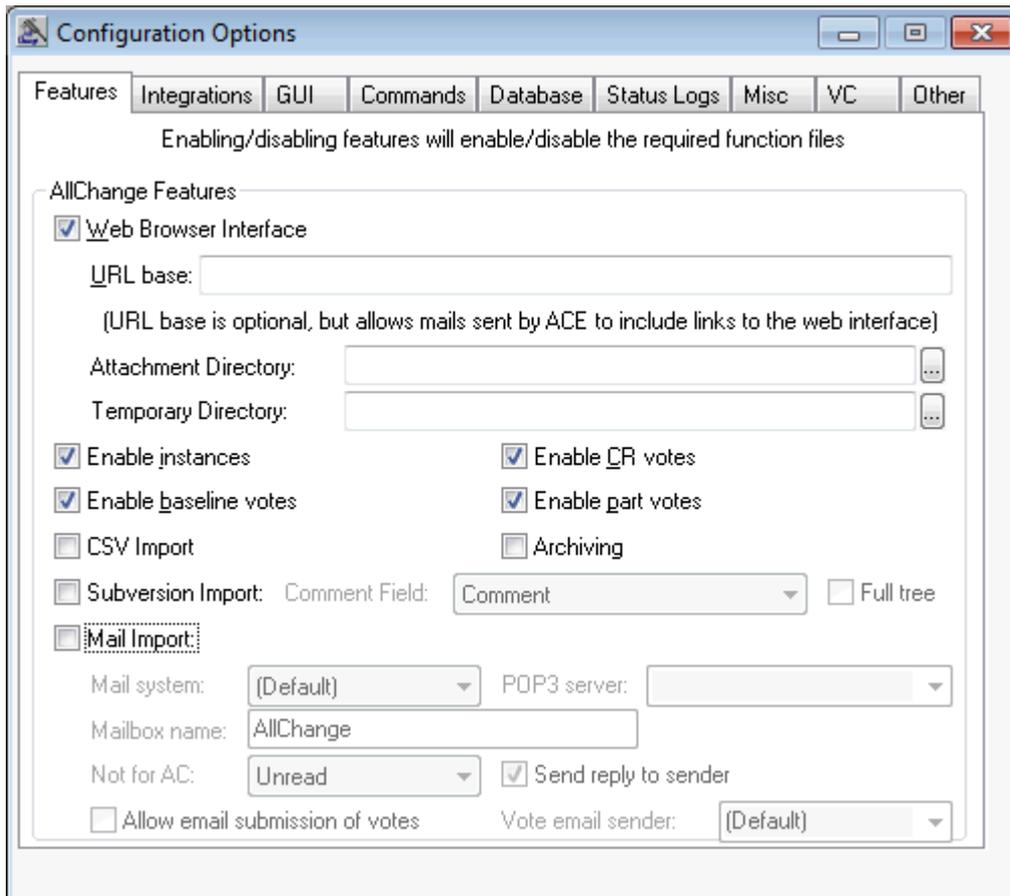
Shows options affecting the **AllChange** version control facilities.

**Other**

Shows site definable options.

## Features

The Features tab shows various options which allow certain **AllChange** features to be enabled/disabled



Enabling/disabling a feature will enable/disable any required [function files](#) for that feature (likewise enabling/disabling a function file will enable/disable the feature corresponding to it).

This provides a simple means of seeing whether a feature is currently enabled, as well as making it straightforward to ensure that the correct function files are enabled to support feature.

It should also be noted that **AllChange** may have additional features that are not listed on the **Features** tab

The features available are:

### Web Browser Interface

Enables the function files needed for the web browser interface to **AllChange** (`webifunc.ac` and `htmlfunc.ac`). Note that in addition to enabling the feature you will also need to perform some installation activities, see [AllChange Web Browser Access](#). The following additional information may also be specified:

#### URL base:

This allows emails sent from ACE to include a link to the web interface. This should be the URL for the **AllChange** web interface in the form `http[s]://<server>[:<port>]/<application>`, e.g. `https://acserver:80/acwebapp`.

#### Attachment Directory:

This may be used to specify an alternative to the default directory to be used when attachments are downloaded - see [Temporary Directories](#).

**Temporary Directory:**

This may be used to specify an alternative to the default directory to be used when attachments are uploaded and for CR, Part and Baseline text- see [Temporary Directories](#).

**Enable instances**

If this is not selected then instances of versions of parts are disabled. All menu items, viewer tabs etc related to instances will not be shown. In addition enabling instances also enables the instfunc.ac function definition file which is used by the out-of-the-box WorkOrder CR class and life-cycle, see [Work Order Change Management Process](#).

**Enable CR votes**

If this is not selected then votes on CRs are disabled. All menu items, viewer tabs etc related to voting for CRs will not be shown. Life cycles with votes defined will behave as if no vote is defined.

**Enable baseline votes**

If this is not selected then votes on baselines are disabled. All menu items, viewer tabs etc related to voting for baselines will not be shown. Life cycles with votes defined will behave as if no vote is defined.

**Enable part votes**

If this is not selected then votes on parts are disabled. All menu items, viewer tabs etc related to voting for parts will not be shown. Life cycles with votes defined will behave as if no vote is defined.

**CSV Import**

Enables the function files needed for the facilities to import parts and CRs from CSV files, csvfunc.ac. See [ACE Facilities for Administrators](#).

**Subversion Import**

Enables the function files needed for the facilities to import parts Subversion, svnfunc.ac. See [ACE Facilities for Administrators](#). In addition the following may be specified:

**Comment Field:**

This allows the field to be used to store the subversion comment for a version. If unspecified then Comment will be used. See Import Parts from Subversion in the reference manual for details.

**Full Tree:**

If checked causes the import from subversion repository browser to load the entire tree from the start. If not checked each folder will be expanded on entry. See Import Parts from Subversion in the reference manual for details.

**Archiving**

Enables the function files needed for the archiving facilities, archfunc.ac. See [ACE Facilities for Administrators](#).

**Mail Import**

Enables the function files needed for the mail import facilities, mailfunc.ac. *Note that Mail import facilities are only supported for named user licenses.* See [Import from Email](#). Mail Import requires the following additional information:

**Mail system**

This specifies the mail system to be used when importing from email, this may be MAPI or POP3. If **(Default)** is selected then the mail system selected on the [Misc](#) options tab will be used (if this is set to SMTP then POP3 will be implied).

**POP3 server**

If the **Mail system** specified is POP3 (either explicitly or as the **(Default)**), then the

POP3 server should be specified here. If **(Default)** is selected then the SMTP Server name/address specified on the [Misc](#) options tab will be used, this is acceptable if the SMTP and POP3 servers have the same name/address.

#### Mail box name

This specifies the mail box to be used when importing from email. The default is the mail box named **AllChange**.

#### Not for AC

This specifies the action to be taken when processing emails which are not directed at **AllChange**. Values may be **Unread** to mark the mail as unread (MAPI only), **Read** to mark the email as read or **Delete** to delete it . If this option is not specified then the mail is marked as unread (MAPI only)

#### Send reply to sender

This specifies whether to send an acknowledgement email to the originator when an import is successfully performed.

#### Allow email submission of votes

If selected then votes may be submitted by email.

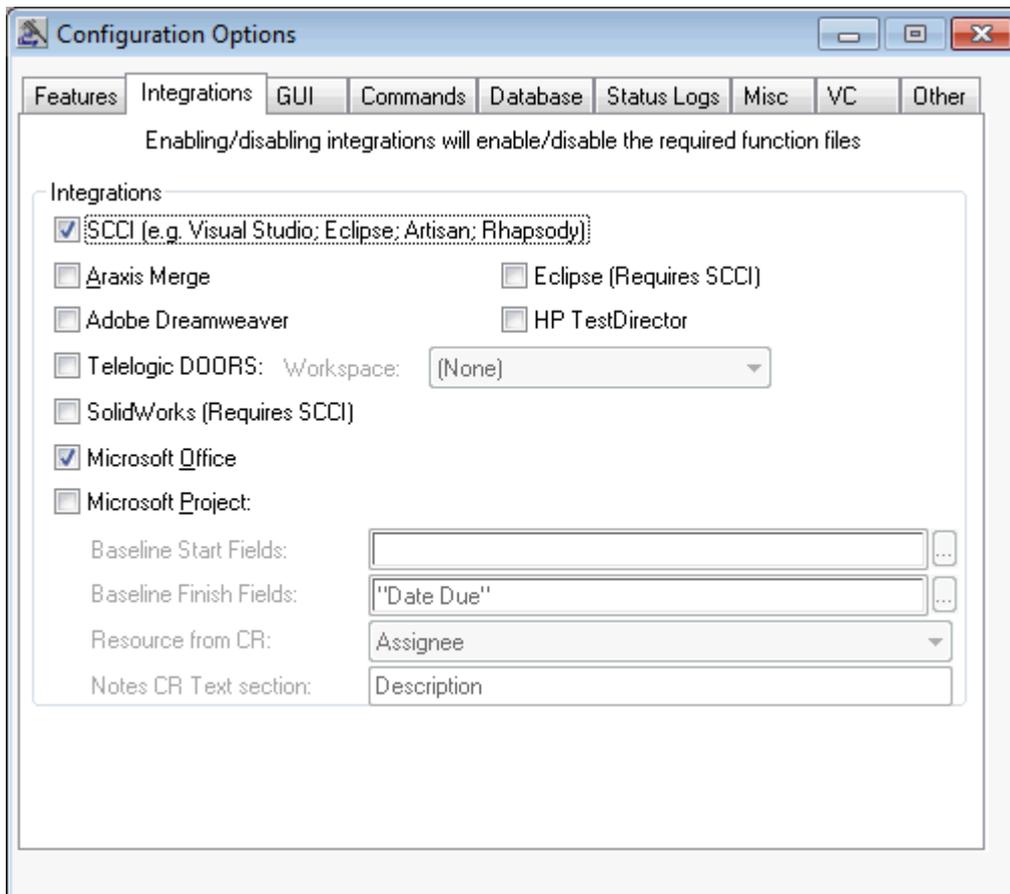
#### Vote email sender

This specifies an **AllChange** user name or email account that is to be used to send the initial vote email from. If an **AllChange** user is specified, then the users' email address specified in the [User Registrations](#) is used. To send from an account that does not correspond to an **AllChange** user, an email address should be specified. If **(Default)** is selected (or empty) then the account specified in the [Send mail from](#) option will be used. If **(Current User)** is selected then the mail will be sent from the current user. *Note that the system used for sending email (e.g. your SMTP or Exchange server) must be configured to allow users to send mails as the specified user.* This is particularly useful if using voting by email since emails may be sent from the email account to which a vote reply needs to be sent.

This option may be overridden by setting the 'Global\_SendMailFromUser' ACCEL variable. This may be useful to temporarily set the sender in ACCEL when sending some types of email from **AllChange**.

## Integrations

The Integrations tab shows various options which allow certain **AllChange** integrations to be enabled/disabled.



Enabling/disabling an integration will enable/disable any required [function files](#) for that feature (likewise enabling/disabling a function file will enable/disable the integration corresponding to it).

This provides a simple means of seeing whether an integration is currently enabled, as well as making it straightforward to ensure that the correct function files are enabled to support integration.

Note that having an integration enabled in the configuration options simply allows the project to be used by the integration. Most integrations will also require other files that support the feature to have been installed on the workstation during a workstation installation.

It should also be noted that **AllChange** may have additional integrations that are not listed on the **Integrations** tab. For instance, the integration with Explorer is only governed by whether the interface was installed on the workstation.

#### **SCCI (e.g. Visual Studio, Eclipse, Artisan...)**

Enables the function file needed for the integration Source Code Control Interface (SCCI), scci-func.ac.. This provides integration with various development tools including, Microsoft Visual Studio, Eclipse, Artisan Real Time Studio, Rhapsody, Access97 and Visual Basic among others. In addition users wishing to use this integration will need to select the **Microsoft source code control interface** option during workstation install. See also [Development Tool Integrations](#) and [Integration with Eclipse](#).

#### **Araxis Merge**

Enables the function file needed for the integration with Araxis Merge, araxfunc.ac. In addition users wishing to use this integration will need to install/enable it for their workstation, see [Integration with Araxis Merge](#).

#### **Eclipse**

Enables the function file needed for the integration with Eclipse, eclipsefunc.ac. In addition users wishing to use this integration will need to install/enable the SCCI integration, see [Integration with Eclipse](#).

**Adobe Dreamweaver**

Enables the function file needed for the integration with Adobe Dreamweaver, dmwvfunc.ac. In addition further set up steps are required, see [Integration with Dreamweaver](#).

**Telelogic DOORS**

Enables the function file needed for the integration with Telelogic DOORS, doorfunc.ac. The workspace to be used for checking modules in and out of AllChange may also be specified, if none is specified then DOORSWS will be used. In addition further AllChange project and DOORS set up steps are required, see [Integration with Telelogic DOORS](#).

**SolidWorks**

Enables the function file needed for the integration with SolidWorks, sldworksfunc.ac. In addition users wishing to use this integration will need to select the SolidWorks Integration during workstation install.

**Microsoft Office**

Enables the function file needed for the integration with Microsoft Office, wordfunc.ac. In addition users wishing to use this integration will need to select the Word/Excel Integration during workstation install.

**HP Testdirector**

Enables the function file needed for the integration with Hp Testdirector, tdvcfunc.ac. In addition further set up steps are required, see [Integration with Testdirector](#).

**Microsoft Project**

Enables the function file needed for the integration with Microsoft Project, mspjfunc.ac. see [Integration with Microsoft Project](#). The data used to map to Project tasks is defined by the following additional configuration options:

**Baseline start fields**

This specifies a list of the names of CR (arbitrary) fields which are to be treated as the *anticipated* start date of the task in MS Project.

**Baseline finish fields**

This specifies a list of the names of CR (arbitrary) fields which are to be treated as the *anticipated* finish date of the task in MS Project.

**Resource from CR**

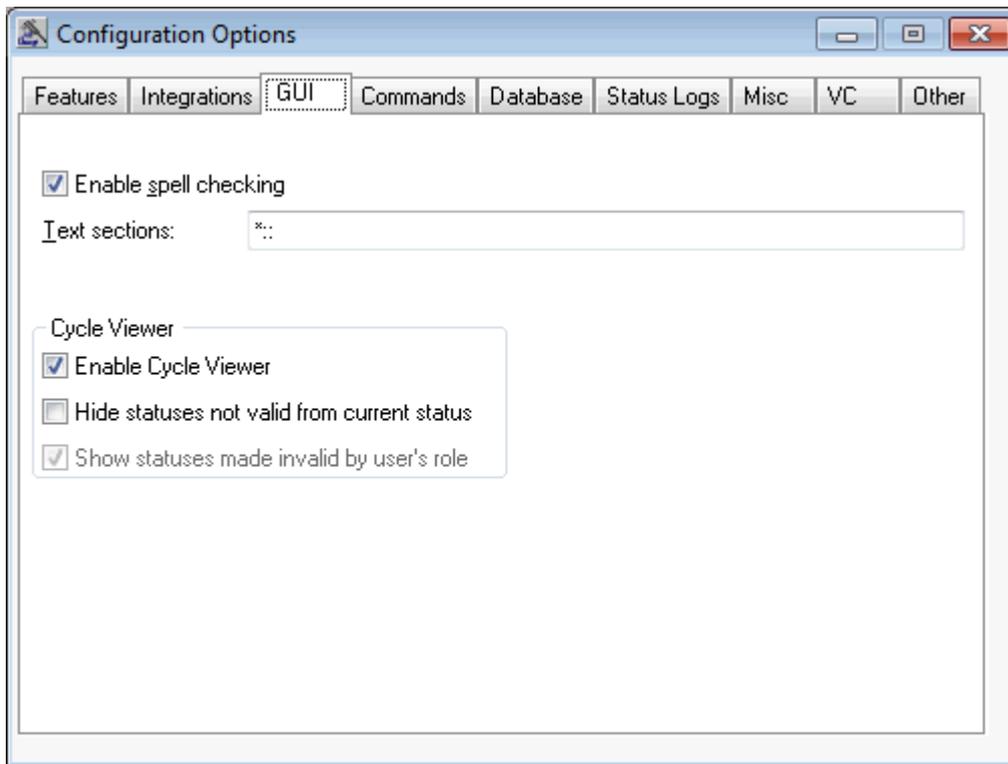
This specifies an ACCEL expression for what to set a task's Resources to in MS Project.

**Notes CR Text section**

This specifies the name of the section in the CR Text which is to be used for the Notes of a task in MS Project.

**GUI**

The GUI tab shows various options which will affect the ACE user interface to **AllChange**.



The options available are:

### Enable spell checking

Enables spell checking in controls that support it. See Spell Checking in the User manual for details.

### Text sections

This entry is used to define the format to be used to recognise sections in the CR, Baseline and Part Text. Each section will be shown in the **Sections** list in the Text tab in the CR, Baseline or Part Viewer in ACE; individual sections may also be extracted into a report. Sections are delimited by placing strings which match a pattern in the CR text; sections must appear on a line of their own. The pattern to be used is specified as the **Text sections**. The pattern is similar to a wild card pattern match: it must contain a single \* (asterisk) where the section header appears; any line matching the pattern is treated as a section header. The default pattern is \*: , so any line ending in : : (double colon) is taken as a section divider with everything up to the : : as the name of the section.

If you prefer a different format for sections — or if you have already created texts which use a different format — you could specify, say, [\*] as the section pattern, in which case a section might look like:

```
[Description of fault]
```

To ease parsing of existing texts which already have additional text on section header lines, the section pattern is actually allowed to have a second \* at the end to denote that there may be further text on the header line: any such text is treated as lying in the section contents. So a pattern of {\*: \* would match:

```
{Description: This is the first line
and this is the second
```

However, future enhancements may require that section headers appear on a line of their own, so it would be preferable not to make use of this facility.

It is important to understand that sections are simply recognisable patterns in the text, so any and all lines matching the pattern delimit the sections. Therefore the pattern should be sufficiently unusual so that it will not occur accidentally in the text; also users must not delete section headers from the text when editing it.

**Enable Cycle Viewer**

It is possible prevent the viewing of life-cycles in the cycle viewer by turning off the "Enable Cycle Viewer" option.

**Hide statuses not valid from current status**

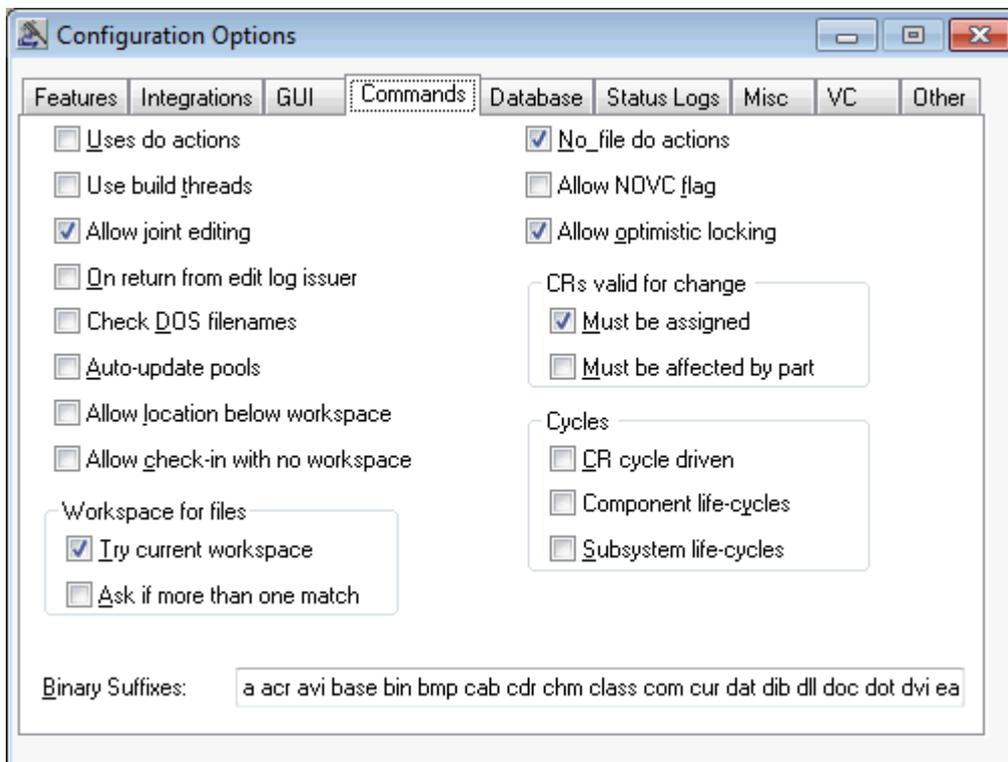
The administrator may disallow the viewing of invalid statuses within the cycle viewer. This overrides any users' preferences.

**Show statuses made invalid by user's role**

Becomes available when the option above is selected. Displays all possible statuses in the cycle viewer regardless of restrictions imposed by the current users role.

**Commands**

The Commands tab shows configuration options that affect the behaviour of *AllChange* commands.



The options available are:

**Uses do actions**

Normally, command actions are not issued when the parts database has a current record which is of type *uses*. This entry causes actions to be executed on *uses*-type parts. See also the *No\_file do actions* entry below.

**No\_file do actions**

Normally, command actions are not issued when the parts database has a current record which has the *no\_file* flag set on it, e.g. the actions of the **copy** command — which are expected to actually copy the part's external file — are not executed for a part which has the *no\_file* flag since it is not expected to have a corresponding file and this simplifies writing the actions. If **No\_file do actions** is selected then (all) actions will be performed on parts with the *no\_file* flag; the command definitions (see [Command Definitions](#)) will have to be modified to test for the flag (via the *pa\_no\_file* field) as required.

**Use build threads**

If this is not selected then build thread (BT) file actions are not performed. This means that BT files, for example, will not be created on issuing a part, will not be copied to pools etc.

If you are using the **AllChange** build facilities then you will need to enable build threads to make full use of the facilities.

#### **Allow NOVC flag**

If selected then the NoVC Flag on parts may be used. This means that parts may be created which are not maintained under version control.

#### **On return from edit log issuer**

This determines the user that is logged as the author of a component version. If this option is *not* selected (the default) then the user who checked in the part will be logged as the author. If this option is selected then the user who did the check out/ issue for edit will be logged as the author.

#### **Allow joint editing**

If selected then parallel developments are to be allowed. This means that the same version of a component may be edited in different workspaces to create a different new version at the same time. If you wish to use optimistic locking for checking in and out then joint editing must be enabled.

#### **Allow optimistic locking**

If selected then the optimistic locking model for parallel development is available to users when checking out for edit

#### **Check DOS filenames**

If selected this will cause any attempt to create a file whose name does not comply with MS-DOS/ Windows 3.x file naming restrictions (i.e. it must have no more than 8 characters to the name plus a 3 character extension) to cause an error to be generated.

#### **Auto-update pools**

If this is selected then on checking in a part from edit, any registered pools will be automatically updated to the new version of the part.

#### **Allow location below workspace**

By default this option is *disabled* which means that whenever a user is attached to a workspace:

- the location field of subsystems from / down to (and including) the workspace part (i.e. the top-level subsystem associated with a workspace, as set in the workspace definition) may be set to anything, **but**
- the location field of **subsystems** below the workspace part is not set to anything (i.e. the location has the default value), **and**
- the location field of **components** below the workspace part is not set to anything (i.e. the location has the default value), or at most it is set to a plain filename (i.e. not a path), e.g. to give a component a different name from a file for descriptive purposes or because the filename contains a character not permitted in a part name.

This option should only be enabled if there are any workspaces whose descendent components require an actual path in their location or whose descendent subsystems are not named the same as their corresponding directory.

Provided this option is disabled certain optimisation are made when converting between file names and part names.

Note however that converting from a filename to a part name when attached to a *non hierarchical* workspace is slow; it *can* also be slow when attached to a *hierarchical* workspace if the parent subsystem that is going to be chosen already has a lot of children and no immediate match is found for the file (such as when files are selected in the File Browser or Explorer which are going to need adding as new components).

This option should not normally be selected.

#### **Allow check-in with no workspace**

If selected then check in new parts or new versions of existing parts which are not in a defined work-

pseudo workspace is created and used for the duration of the checkin associating it with a sub-system which is prompted for on check in.

#### Workspace for files:

##### Try current workspace

If selected then when AllChange maps from a file name to a part name (e.g. when checking in a file from the file browser or Windows explorer), then if the current workspace is valid for the file being checked in, this will be taken to be the workspace to use. This is useful when there are workspaces with overlapping directories. This should be avoided wherever possible and is strongly discouraged.

##### Ask if more than one match

If selected then if more than one workspace matches a file when mapping from a file name to a part name, then the user is prompted to select which workspace is to be used.

#### CRs Valid for Change:

##### Must be assigned

If selected then CRs which are *valid for change* (i.e. may be used to authorise parts being checked out) must be assigned to the user performing the check out.

Note that if this is *not* selected then it is important to ensure that the **Command Access** rules allow non cr assignees to modify a CR as this will be required when a part is checked back in which was checked out against a CR.

##### Must be affected by part

If selected then CRs which are *valid for change* (i.e. may be used to authorise parts being checked out) must have the part being checked out as a part affected in order to authorise a change to that part.

#### CR cycle driven

If selected then parts may only be edited (i.e. checked out for edit) via a life-cycle status change which is invoked via a CR life-cycle status change. This should be used if your CR life-cycle causes all the parts affected by the CR to have their status changed and this in turn is to cause a new version to be created.

#### Component life-cycles

If selected then life-cycles on components may be used rather than versions. This will cause the status of a component to be shown on the **Part** tab of the part viewer in ACE. This should only be used if branching of versions is *never* to be used.

#### Subsystem life-cycles

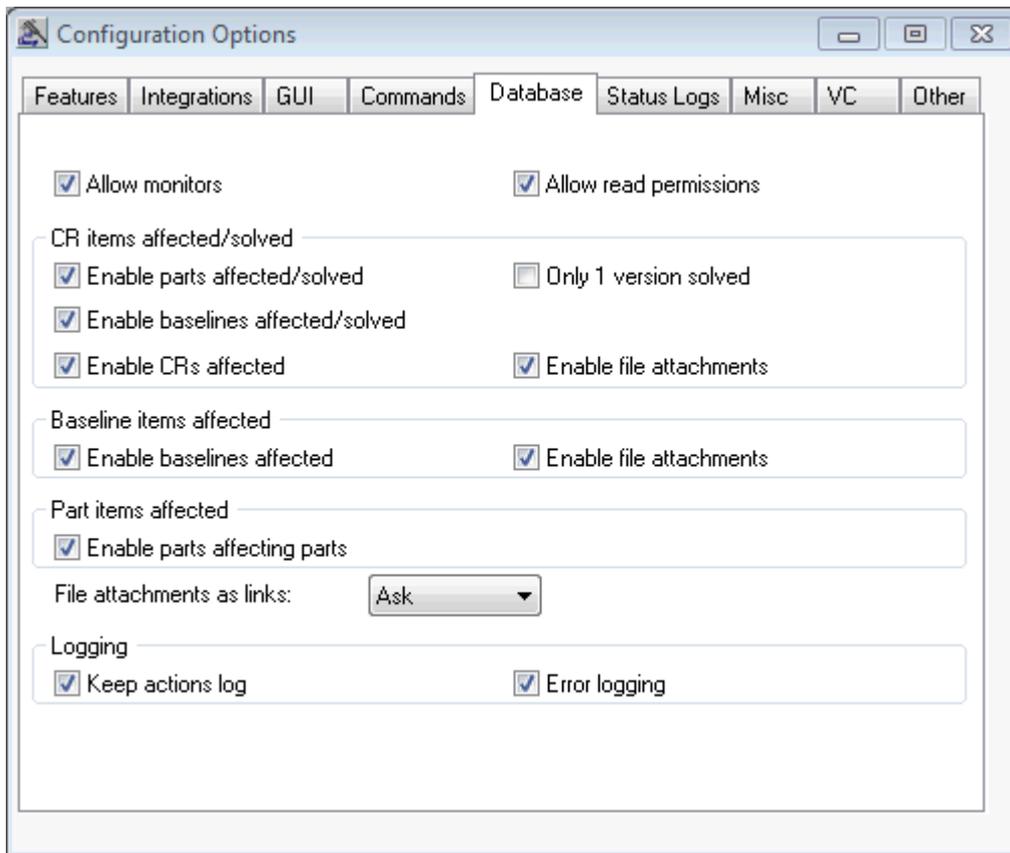
If selected then life-cycles on subsystems may be used. This will cause the status of a subsystem to be shown on the **Part** tab of the part viewer in ACE.

#### Binary Suffixes

This specifies a space separated list of the file name suffixes which are to be regarded as binary files. As a special case . (period) may be used to denote files with no extension. This will effect the flags passed on to the VC tools for how to handle the file.

#### Database

Shows configuration options affecting the information stored in the database. There may be some data that your project or site does not wish to use and therefore it should be disabled to simplify the user interface and improve performance in some cases.



The options available are:

**Allow monitors**

If this is not selected, then monitoring is disabled. This means that *AllChange* will not search for any monitors for any events. This may lead to a small improvement in performance if monitors are not required.

**Allow read permissions**

If this is not selected, then checking for read permissions is disabled. This means that all data is readable/viewable by all users. This may be switched off if read permissions are not used as it is a performance overhead

**Enable parts affected/ solved**

If selected enables information to be kept in the database about the relationship between CRs and the parts that they affect and the versions that they solve. If not selected this information may not be stored and the ACE view windows for parts and CRs will be modified accordingly.

**Only 1 version solved**

This specifies that a CR may only have one version of a part associated with it as a version solved. Multiple versions might be required, for example, to allow a single CR to be used for a change in several lines of development. However, if it is a rule that a CR should be raised for each line of development then you would probably wish to enforce that only one version solved could be associated with a CR in which case this option should be selected.

**Enable baselines affected/ solved**

If selected enables information to be kept in the database about the relationship between CRs and the baselines that they affect and the baselines containing the implementation of the CR. If not selected this information may not be stored and the ACE view windows for baselines and CRs will be modified accordingly.

**Enable CRs affected**

If selected enables information to be kept in the database about the relationship between CRs and

other CRs. This information may be used to implement a hierarchy of CRs. If not selected this information may not be stored and the ACE view windows for CRs will be modified accordingly.

#### **Enable file attachments (CR)**

If selected allows the attachment of files to CRs. If not selected this information may not be stored and the ACE view windows for CRs will be modified accordingly.

#### **Enable baselines affected**

If selected enables information to be kept in the database about relationships between baselines and other baselines. This allows informational information about the relationships between baselines to be maintained, note that this is distinct from the meta-baseline relationship to the baselines it contains. If not selected this information may not be stored and the ACE view windows for baselines will be modified accordingly.

#### **Enable file attachments (Baseline)**

If selected allows the attachment of files to baselines. If not selected this information may not be stored and the ACE view windows for baselines will be modified accordingly.

#### **Enable parts affecting parts**

If selected allows a part to be associated with other parts as *parts affected*. If not selected this information may not be stored and the ACE view windows for parts will be modified accordingly.

#### **File attachments as links**

This specifies how baseline and CR file attachments should be handled. Attachments may be treated as *links* to existing files, in which case **AllChange** will simply refer to the attached file, or *copies* of the attach file in which case **AllChange** will take a copy of the attachment into a secure area. Values may be:

**Ask:** prompt for each attachments whether to hold a link

**Never:** links are not to be used, a copy will always be taken

**Always:** links will always be used

#### **Keep actions log**

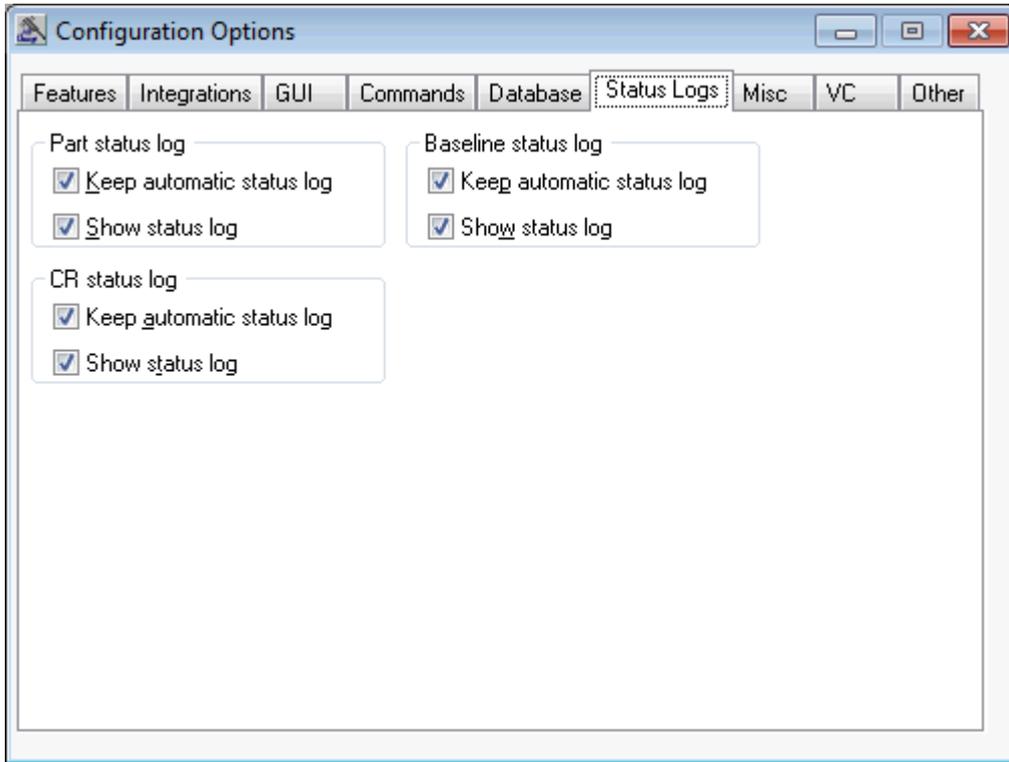
If selected then all actions performed by users will be logged in a file called `acacts.log` in the **AllChange** project directory. `ACCONFIG` will create a new log file when this option is selected and the project saved. If the file already exists then it will be saved as `acacts.bak`.

#### **Error logging**

If action logging is enabled, then error logging may also be enabled. This will cause all errors to be logged in the actions log file `acacts.log`. Fatal errors will include the string `ERROR :`, warnings will include the string `WARNING :`.

#### **Status Logs**

Shows information affecting the status log information maintained by **AllChange**.



**Keep automatic part status log**

**Keep automatic baseline status log**

**Keep automatic CR status log**

Automatic status logging — the creation of a record in the status logs database every time the status of a part, baseline or CR is changed — is enabled separately for each of the three databases. To disable status logging for a database one or more of these entries should not be selected. It would be advisable to disable status logging for those databases for which the historical information is not required — the log is liable to get large! Even if automatic status logging is disabled this does not preclude the use of the ACCEL `add_status_log` function in the command definitions or cycles database actions to explicitly log certain selected events of interest. The default is to keep status logs all three databases.

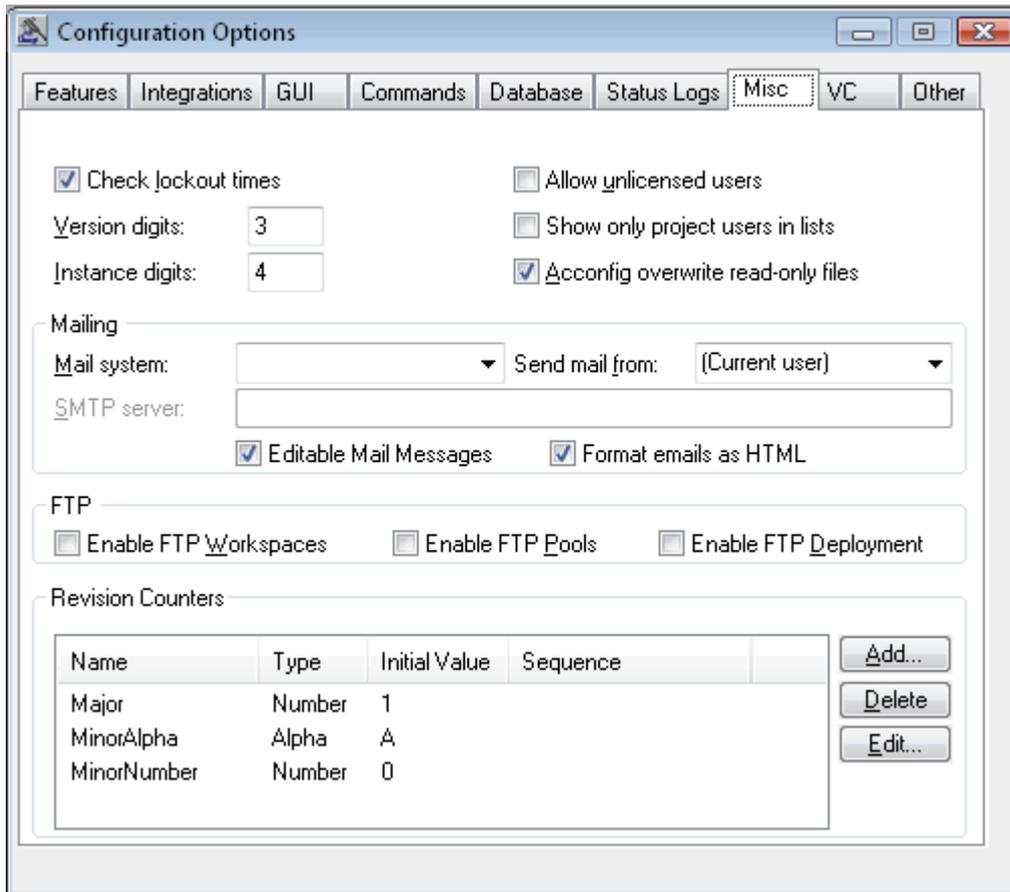
**Show part status log**

**Show baseline status log**

**Show CR status log**

These options allow you to control whether the status log tab on the part, baseline and CR viewer windows is shown. This is independent of whether automatic status logging is enabled.

## Misc

**Allow unlicensed users**

If this is selected then unlicensed users will be permitted read-only access to the **AllChange** database.

**Check lockout times**

If selected then **AllChange** will check lockout times from the lockout file and issue warnings or log users off as appropriate, see [Lockout Times](#).

**Show only project users in lists**

If selected then only users permissible for the current project are shown in lists of users in ACE (e.g. The list of possible CR assignees), otherwise all registered users for all projects are shown. The permissible users for a project are specified in the [project definition](#).

**ACONFIG overwrite read-only files**

If selected this option prevents ACONFIG from overwriting configuration files which are read-only. This is useful when you are using **AllChange** to control the configuration file for your **AllChange** projects as it ensures that files which are checked out read-only are not modified until they have been checked out for edit.

**Version digits**

This causes the number of digits used in constructing a new version to be set to a fixed number. New versions will have their version-id number padded with leading zeroes to this number of digits, e.g. `component;001` if `num-digits` is 3. If this empty/ blank then the number of digits is variable (e.g. 1, 2, 10, 11).

**Instance digits**

This causes the number of digits used in constructing a new instance to be set to a fixed number.

New instances will have their id number padded with leading zeroes to this number of digits, e.g. `component;001:00001` if `num-digits` is 4. If this empty/ blank then the number of digits is variable (e.g. 1, 2, 10, 11).

This feature is useful to make the database store versions in true ascending order of version number; lists of versions will then automatically be shown to the user in this order. (Without this the database places `component;9` after `component;10`.) References to the version will always have to include any leading zeroes. The number of digits chosen should allow for the highest version number which any part will ever have. If the number of version digits is changed then all the existing versions should be renamed to have the new number of digits.

### Mail system

This option allows you to select the mail system that you want **AllChange** to use to keep users informed of events occurring, for example, when a CR is assigned to a user.

Mail systems supported are:

**LotusMail:** this should be used for Lotus and CC mail and any other `VIM` compliant mail systems

**MAPI:** this should be used for Microsoft Mail, Outlook, Exchange and any other `MAPI` compliant mail systems

**SMTP:** this should be used for UNIX mail and other `SMTP` compliant mail systems

**None:** this should be used if you wish to disable mailing in ACE.

Note that if you are using the Web Browser interface and you wish email notification to function from this then you must use **SMTP**.

See [Integration with Mail](#) for further details of the implementation of the mail system integrations.

### Send mail from

This specifies an **AllChange** user name or email account that from which all emails sent by **AllChange** should be sent. If an **AllChange** user is specified, then the users' email address specified in the [User Registrations](#) is used. To send from an account that does not correspond to an **AllChange** user, an email address should be specified. If (**Current User**) is selected then the mail will be sent from the current user. *Note that the system used for sending email (e.g. your SMTP or Exchange server) must be configured to allow users to send mails as the specified user unless (Current User) is to be used.*

This option may be overridden by setting the 'Global\_SendMailFromUser' ACCEL variable. This may be useful to temporarily set the sender in ACCEL when sending some types of email from **AllChange**.

### SMTP server

This allows the name of your SMTP server to be defined if you have selected **SMTP Mail system**. If no value is specified then the address of the *sender* will be used to determine what SMTP server to use (which may or may not be correct).

### EditableMailMessages

This specifies whether mail messages can be altered before sending. If enabled, any mail messages being automatically sent by **AllChange** will prompt the user to alter the message before sending. The default value is True, set it to False to disable editable email messages.

### Format emails as HTML

This enables emails sent by **AllChange** to be sent in HTML format. This allows emails to be formatted to include HTML features such as hyperlinks, tables, etc. This is only available if using SMTP or MAPI mail and **AllChange** will issue an error message if attempting to send HTML messages with any other mail system. See [About Integration with Mail](#) for further details.

### Enable FTP Workspaces

If selected this enables the use of FTP to and from workspaces which refer to directories on remote platforms, see [Workspaces](#) for details.

### Enable FTP Deployment

If selected this enables the use of FTP to deploy HTML documents to remote Web server platforms, see [Workspaces](#) for details.

### Enable FTP Pools

If selected this enables the use of FTP to pools which refer to directories on remote platforms, see [Pools](#) for details.

### Revision Counters

Allows the definition of revision counters to be used in [field name definition attributes](#) to define a complete revision number.

Each Revision Counter has the following attributes:

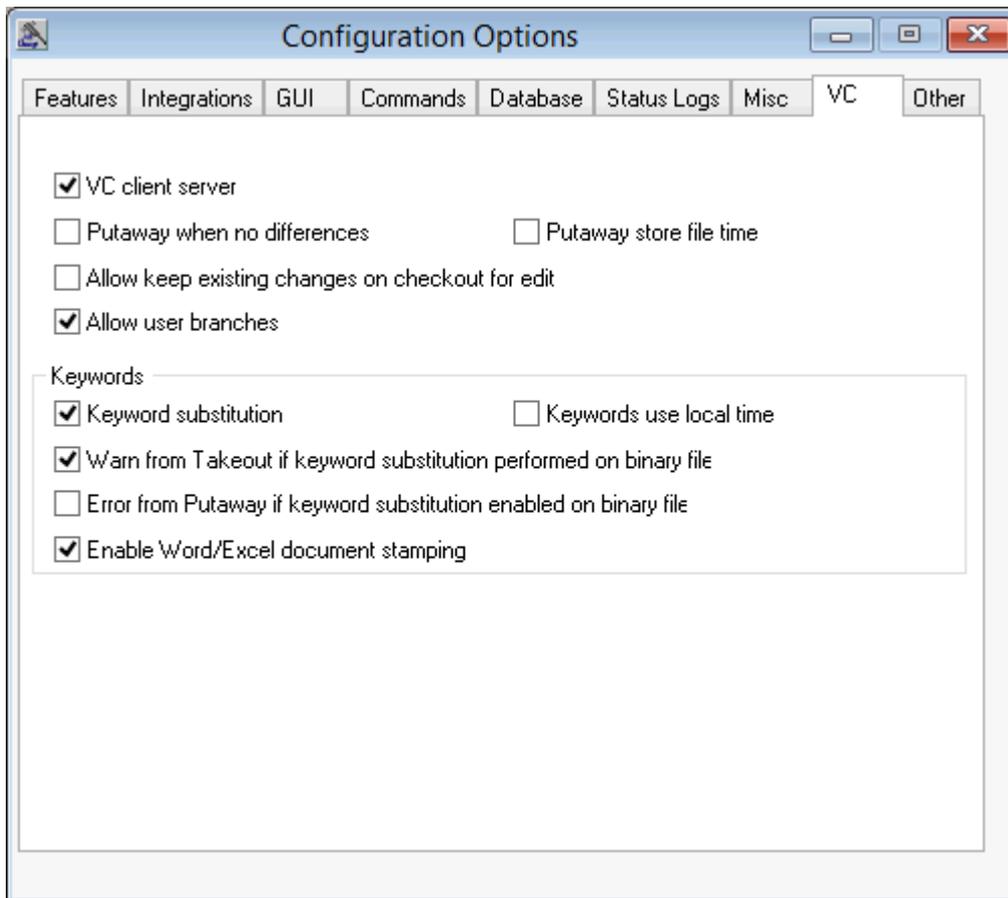
- **Name:** the name of the counter. This may not contain spaces.
- **Type:** the type of the counter. This may be:
  - **Alpha:** A sequence of single characters, if no value sequence is specified the default is A-Z
  - **Number:** An integer number
  - **List:** A user defined list of values
- **Initial Value:** the initial value to be used and whenever the counter is reset
- **Value Sequence:** this may be specified for Alpha or List Types. For Alpha type this must be a sequence of single characters. For List type this should be a space separated list of values, values may not contain spaces.

For example in the Intasoft Standard configuration there are 3 revision counters defined:

- **Major** - this is a numeric counter - this is used in the [Issue No](#) and [Revision](#) version arbitrary fields
- **MinorAlpha** - this is an Alpha counter using the default values A-Z - this is used in the Revision version arbitrary field
- **MinorNumber** - this is another numeric counter - this is used in the Issue No version arbitrary field.

### VC

The VC tab shows configuration options which affect the version control facilities within **AllChange**.



The options available are:

#### VC client server

If selected this will cause the version control files (VC files) in **AllChange** to be accessed in client/server mode of operation. This option has no effect unless **AllChange** is being run itself in client/ server mode of operation. See [Client/ Server](#) for full details.

#### Putaway when no differences

If selected then, when a part is *checked in from edit* a new version of the part will be created even if no changes have been made.

#### Putaway store file time

This will cause the VC tool **Putaway** (which actually stores the contents of the versions for any files stored under **AllChange**'s version control) to store (and restore) the time stamp that the file had when the version was stored (**Checked In from edit**). Without this option the time that the **Check In** from edit took place will be stored. This time is also shown on the part version and is the time stamp that workfiles created on checking out the version will have.

#### Allow keep existing changes on checkout for edit

Normally, if the user is performing a **CheckOut for Edit** and **AllChange** finds that a component is currently checked out read-only to the workspace it will (silently) replace the corresponding workfile with the contents of the version being checked out for edit (since a version checked out read-only ought not to have been changed). If at your site you are concerned that users might have begun their edits/ changes while *the same version* was checked out read-only you may set this option: whenever this situation arises ACE will compare the contents of the workfile against the version to be checked out for edit and, if they differ, will ask the user whether the current file's contents should be preserved during the checkout for edit. Note that this operation can be quite slow, so you should only enable this option if you find it necessary.

#### Allow user branches

If this is selected then users may create branches with any name. If this is not selected then branch

names must be selected from a defined list which may be maintained by the AllChange administrator or users with permission to do this.

**Keyword substitution**

If selected then files are scanned for version control keywords which are replaced by the appropriate information when a file is checked out. Key word substitutions will not be performed on files which are specified as *binary*, i.e. files which have a suffix identified in the **Binary Suffixes** option on the **Commands** tab of the configuration options.

**Keywords use local time**

If selected then any keyword substitutions or Word document stamping of any date or date/time keywords will be in the current user's local time. Without the option enabled, the keywords will be formatted as UTC. The default is disabled. (Note that this will cause users in different time-zones viewing a local workfile for the same part to see different times and/or dates).

**Warn from Takeout if keyword substitution performed on binary file**

If selected then when checking out, if the content seems to be binary and a keyword substitution is found and performed, a warning is issued at the end. This option is *enabled* by default. The check out succeeds, but the user should verify that all is well. The test for whether content seems to be binary is simply whether it contains any byte with value 0. This is a simple criterion, but one used by other well-known applications.

**Error from Putaway if keyword substitution enabled on binary file**

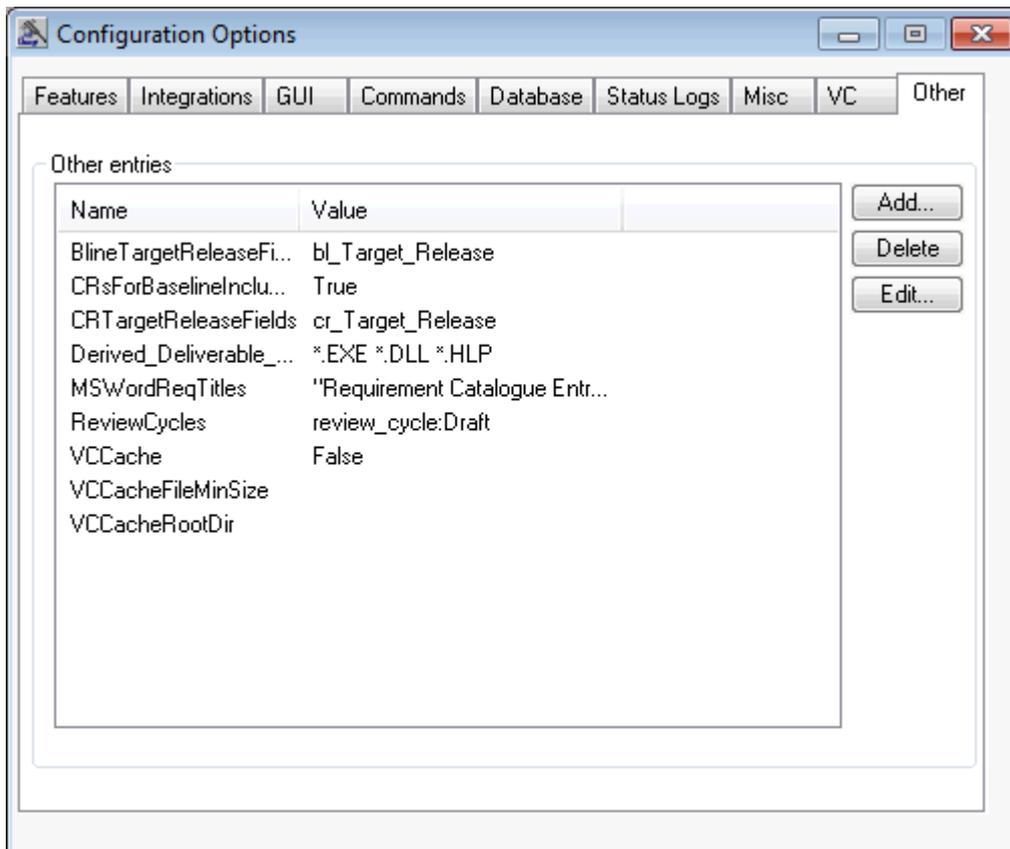
If selected then when checking in, if the (new) content seems to be binary - regardless of whether any keywords might actually be found at check out - an error is raised. This option is *disabled* by default. If enabled, the check in would fail. The solution would generally be to add the suffix for the file to the list of Binary Suffixes. Use this to prevent files with unintended substitutions being accepted at all. The test for whether content seems to be binary is simply whether it contains any byte with value 0. This is a simple criterion, but one used by other well-known applications.

**Enable Word document stamping**

If selected this enables the automatic stamping of word documents with keywords when a part is returned from edit or checked in, see [Integration with Microsoft Word](#) for details.

**Other**

The Other tab shows configuration options which may be site specific. New options may be added for use in ACCEL code.



The **Add**, **Delete** and **Edit** button may be used to add new options, remove options and modify the content of options.

As supplied there are some **Other** configuration options which are used for features such as the release management process and integrations.

These are:

#### **BlineTargetReleaseFields**

This specifies the fields which may be used to specify the **Target Release** for a baseline. Each field should be specified as an ACCEL field reference.

#### **CRTargetReleaseFields**

This specifies the fields which may be used to specify the **Target Release** for a CR. Each field should be specified as an ACCEL field reference.

#### **CRsForBaselineIncludeLinks**

This specifies whether the **CRs For Baseline** function should automatically include all related CRs

#### **Derived\_Deliverables\_Patterns**

This specifies a space separated list of the patterns which match derived deliverable files. This is used by the `sw_release_cycle` when promoting derived files which are deliverable to the system test area.

#### **ReviewCycles**

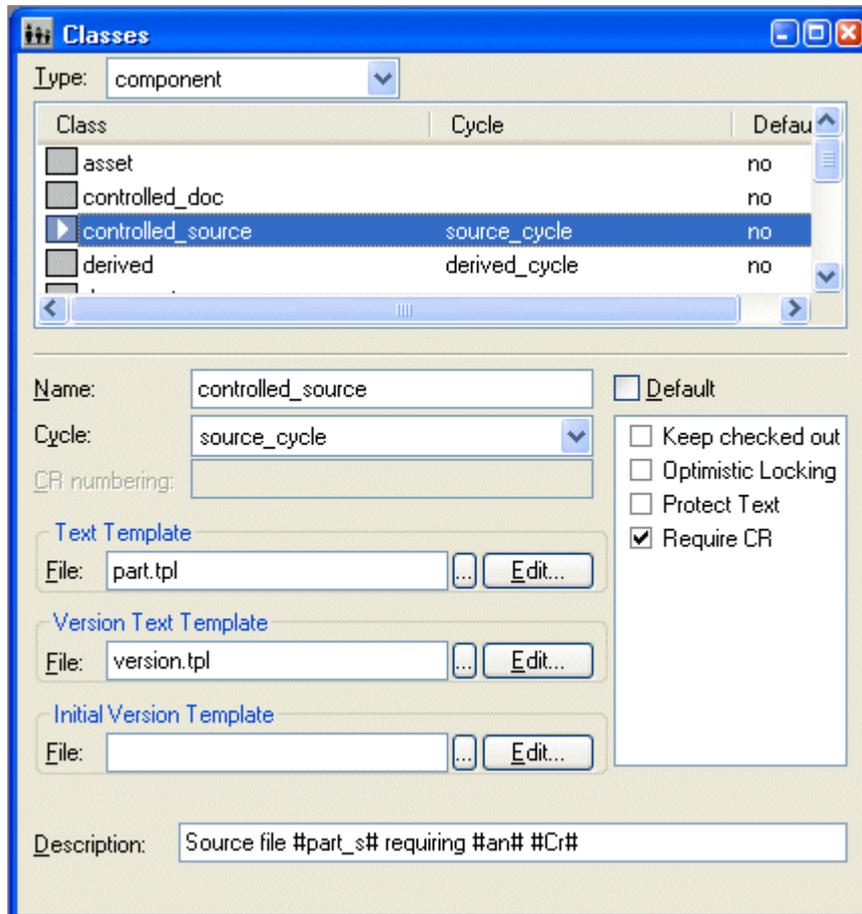
This specifies life-cycles which are to be regarded as having a review type of cycle and which status in the cycle is to be regarded as a 'draft' status. This should be specified in the form of a space separated list of *cycle-name:draft-status*, for example `review_cycle:Draft`.

## **Classes**

## About Classes

Parts, CRs and Baselines may all have a class associated with them which determines various attributes of items of that class such as the life-cycle, the arbitrary fields used and in the case of CRs the CR "numbering" scheme.

From within ACCONFIG the list of currently defined classes may be viewed and modified from **Plan | Classes**.



The classes are grouped by the **Type** of item the class pertains to. It must be one of the following:

<b>component</b>	only components may have this class
<b>subsystem</b>	only subsystems may have this class
<b>cr</b>	only CRs may have this class
<b>baseline</b>	only baselines may have this class

Each class has the following information:

### Name

An arbitrary name for a class (e.g. *Source*, *Document*, *hardware*, *cr*). The name must be comprised of alphanumeric characters and the `_` character and may not include a space character. This is used to classify the item, e.g. source code, documentation or hardware for a part, bug fix or upgrade request for a CR, test or release for a baseline. This can be used for searching and reporting purposes, to restrict what operations may be performed on items, and in conditions to other commands so that, for example, only parts of a certain class are included in a baseline.

### Default

This option determines whether the class is to be the default class for its **Type**. The default class will

be selected as the default value when new items of that type are created in ACE. If no default is specified then the first class (alphabetically) will be selected.

**Cycle**

Specifies the life-cycle, if any, that items of the class will have. The life-cycles are defined in the cycle definitions of the configuration management plan, see [Life-cycles](#). Component parts which do not have a life-cycle will be checked out for edit/checked in using the **Issue/Return** commands, whereas those which have a life-cycle will be checked out for edit/checked in using the life-cycle.

**Attributes**

Different attributes may be specified depending on the **Type** of the class as shown in the table below.

Attribute	Type	Description
Dreamweaver Integration	any	Specifies that components of this class are used <i>only</i> with the Dreamweaver Integration. If the Integration is not enabled then this attribute will not be available for selection, See <a href="#">Configuration Options, Integrations</a> tab. Likewise any class defined with this attribute when the integration is not enabled will not be shown in ACE.  In addition any fields defined for this class only will only be shown in <a href="#">Field Name Definitions</a> and in ACE if the Integration is enabled.
Eclipse Integration	any	Specifies that components of this class are used <i>only</i> with the Eclipse Integration. . If the Integration is not enabled then this attribute will not be available for selection, See <a href="#">Configuration Options, Integrations</a> tab. Likewise any class defined with this attribute when the integration is not enabled will not be shown in ACE.  In addition any fields defined for this class only will only be shown in <a href="#">Field Name Definitions</a> and in ACE if the Integration is enabled.
Keep checked out	component	Specifies that components of this class should be kept checked out read only. This means that when a component is checked in from edit, it will automatically be checked out again read only
Microsoft Office Integration	any	Specifies that components of this class are used <i>only</i> with the Miscrosoft Office Integration. . If the Integration is not enabled then this attribute will not be available for selection, See <a href="#">Configuration Options, Integrations</a> tab. Likewise any class defined with this attribute when the integration is not enabled will not be shown in ACE.  In addition any fields defined for this class only will only be shown in <a href="#">Field Name Definitions</a> and in ACE if the Integration is enabled.

Microsoft Project Integration	any	Specifies that components of this class are used <i>only</i> with the Microsoft Project Integration. . If the Integration is not enabled then this attribute will not be available for selection, See <a href="#">Configuration Options, Integrations</a> tab. Likewise any class defined with this attribute when the integration is not enabled will not be shown in ACE.  In addition any fields defined for this class only will only be shown in <a href="#">Field Name Definitions</a> and in ACE if the Integration is enabled.
No File	component	Specifies that components of this class are have the No File flag set on creation (i.e. there is to be no file associated with components of this class)
Optimistic Locking	component	Specifies that components of this class are checked out using optimistic locking by default
Protect Text	any	Specifies that items of this class should not allow users to modify the text field after initial creation, but allow addition to sections with a date/time/user stamp to provide an audit trail
Require CR	component	Specifies that components of this class require a CR to be specified authorising any changes. This comes into effect when a component is checked out for edit, a new component is added or if a newversion of a component is checked in without a preceding checkout.
SCCI Integration	any	Specifies that components of this class are used <i>only</i> with the SCCI Integration. . If the Integration is not enabled then this attribute will not be available for selection, See <a href="#">Configuration Options, Integrations</a> tab. Likewise any class defined with this attribute when the integration is not enabled will not be shown in ACE.  In addition any fields defined for this class only will only be shown in <a href="#">Field Name Definitions</a> and in ACE if the Integration is enabled.

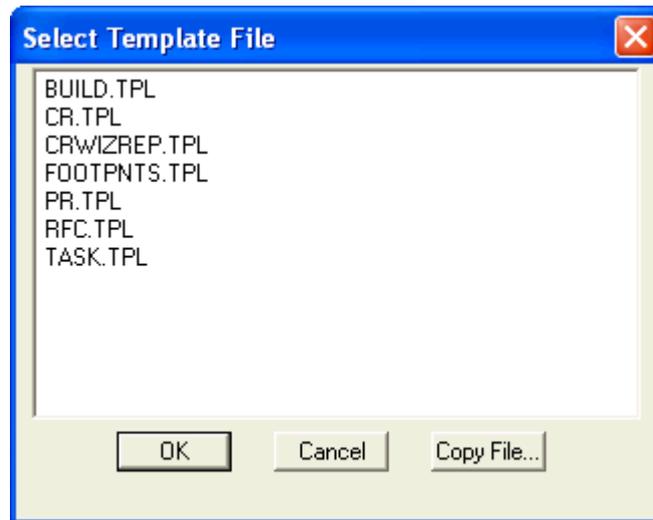
### CR numbering

This is only applicable to classes of **Type** `CR`. It determines how CRs of this class are numbered as described in CR Numbering. If no numbering is specified then the default CR numbering scheme will be used (this is a simple 5 digit number).

### Text Template

This is applicable to classes of any **Type**. It determines the name of the file to be used as the template (i.e. initial contents) for the texts of new items of this class.

The template file may be selected using the ... button. The file selection dialog will show a list of existing template files, if a new template file is required which exists in another directory this may be copied to the appropriate template directory using the **CopyFile** button.



### Version Text Template

For a component class, this determines the text template to use for the initial contents of version texts.

### Initial Version Template

For a component class, this specifies a template file to be used as the contents of the initial version of a part when created with this class.

### Specifying template files

For text templates, if a brand new file is to be created the name of the file may be entered in the **File** edit control and on selecting the **Edit** button, will be created in the appropriate template directory and the editor invoked for it.

The template will be sought in the **AllChange** project directory and then in the system directory. Different classes of item may share the same template file. The file may be created, if it does not exist, or modified if it does using the **Edit...** button.

For initial version templates:

- The template files are sought in a subdirectory of the project or system directory called `template`
- If a workfile exists when the component is created this is used instead of the template file for the first version
- If no template file exists and there is no workfile then an empty first version will be created

For text templates:

- A path may be specified for the template file in which case the template file will be sought in that directory alone
- If no **text template** is specified then a default template will be used. For CR classes this is `cr.tpl`, for baselines it is `baseline.tpl`, for parts it is `part.tpl` and for versions it is `version.tpl`.
- A template file *must* exist for a new item to be created;
- When editing a text template the supplied text editor ACEDIT will be invoked. Template files may contain whatever is desired (including being empty), for example it may contain various sections which are to be completed - these sections will be recognised by ACE if they match the pattern specified in the **Text Sections** configuration option, see [Configuration Options](#).

**Description**

Specifies a description to be shown as a tool tip when the mouse hovers over the class in a viewer or in a drop down list of classes. Nomenclature mapping is supported

The classes definitions are stored in the configuration file `classes.ac` in the project or system directory. It must be present for the class and life-cycle aspects of **AllChange** to function.

**CR Numbering**

The way in which CRs are *numbered* — which is how they are referred to within the system — is site-specifiable, and may be made to vary according to the CRs class. A CRs "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.

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.

What format and which counter to use when generating a new CR number are determined by the new CRs class. The **CR numbering** is specified as a string as follows:

1. a % (percent) character followed by a digit (0–9) is replaced by (the next value of) the corresponding auto-incremented counter (i.e. a system-generated number) — the format is 5 digits (with leading zeroes);
2. a % character followed by an i character is replaced by the value of the (optional) **ID** option to **newcr** (i.e. a completely user-supplied string);
3. all other characters are copied without change into the new CR number (i.e. fixed text) — no spaces are allowed.

Examples:

<code>%0</code>	5 digit number using counter 0, e.g. 00023 (this is the default if no CR number format is associated with the CRs class).
<code>%i</code>	CR number to be completely supplied by user when creating a new CR (e.g. <code>External#1234</code> ); no auto-incrementing counter used.
<code>BUG-%2-%i</code>	CR number uses fixed text, auto-incrementing counter 2 and a suffix supplied by the user, e.g. <code>BUG-54321-jon</code> .

CRs are sorted and indexed lexically by the (whole of the) CR number string; upper- and lower-case are *not* the same. When using a number originating outside of **AllChange** leading zeroes may be desirable, otherwise `MyCR-234` will appear after `MyCR-1234`. When deciding on a numbering scheme it should be borne in mind that searches of the database and displays in ACE can be limited to left-hand side matches of the CR number string, so grouping related CR types by common prefixes (e.g. `BUG-` or `PROJ1-`) is a good idea.

Each counter is incremented and used quite independently of the others. Each counter produces numbers in the range 00000–99999. More than one CR class may stipulate that it uses the same CR counter if that is desired by using the same %*digit* in the format specification; the rest of the format may be identical to or differ from the other CRs.

The CR number format is used *only* when creating a new CR of a certain class; once the CR has been created there is no link between the CRs number and the class' CR number format (sites which allow

different CR classes with different CR numbering may wish to prevent a CRs class from being altered). The system automatically numbers a new CR one higher than the last CR. (Note that it is possible to create "gaps" in CR numbering where CRs have been deleted; this has no adverse effects.) If the numbering format is changed once there are CRs of that class, new CRs will bear the new numbering; it may be desirable to change existing CRs over to the new format. CR counters may be reset using the ACCEL function `set_crnum_resource_value()` only; a used counter could be reset to 0 and used for new CRs of a new class if no new CRs are to be created of the class currently using the counter.

CR numbers (as a whole) must be unique. For speed the system assumes a new CR number generated from an auto-incrementing counter will be unique (an error will arise if it is not); if the format uses no counter (e.g. just `%i` — totally user-supplied) an error is issued if the number already exists. An error is issued if the user supplies an **ID** for a new CR whose format does not include a `%i`. The entry condition for **newcr** may implement checks on the user-supplied ID.

A CR number should not begin with a `#`, `/` or `!`; it should not contain a `'`, `"` or (space). To allow for future developments it would be preferable not to start a CR number with a punctuation character and to use only `_`, `-`, `:` and `.` punctuation characters.

The name of the file in which the CR text is placed/ expected by **getcrtext**, **editcrtext** and **putcrtext** is basically the same as the CR number, except that any punctuation characters are removed, the string `cr` is prepended to the name if the name starts with a digit, e.g.:

```
01234           → cr01234
My/CR-1234:ABC → MyCR1234ABC
```

When deciding on a format for a CR class this should be borne in mind. If CR numbers are designed such that different numbers can produce the same text filename then entries in the command definitions file may have to be changed to recognise this (e.g. refuse to extract a CR if a matching CR text filename already exists). An ACCELfunction, `cr_text_filename`, returns this filename for a specified CR number.

## Life-cycles

Life-cycles may be defined which are used by parts, CRs and baselines to implement development and change control procedures. A life-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, implement approval procedures, log progress etc.

The class of a part, CR or baseline determines the life-cycle of an item so you can, for example, have a different life-cycle for different classes of part (e.g. source code and documentation) and a different life-cycle for different classes of CR (e.g. Error, Problem Report).

Each life-cycle consists of a **Name** and a set of **Statuses**. The statuses are the stages of the cycle: as an item moves from one status to another it is moving through its life-cycle.

Details of all status changes are logged in the status logs database (unless automatic status logging has been disabled). This provides an historical record of the progress undergone by an item.

In the case of parts, subsystems and components may have their own life-cycles; versions have the life-cycle of the component. Subsystems, components and versions all have their own statuses so, for example, different versions of a component may be on different statuses and move through the life-cycle independently thus allowing for variants and parallel developments. The life-cycle may be used to create new versions of components and this should be considered when developing the life-cycle.

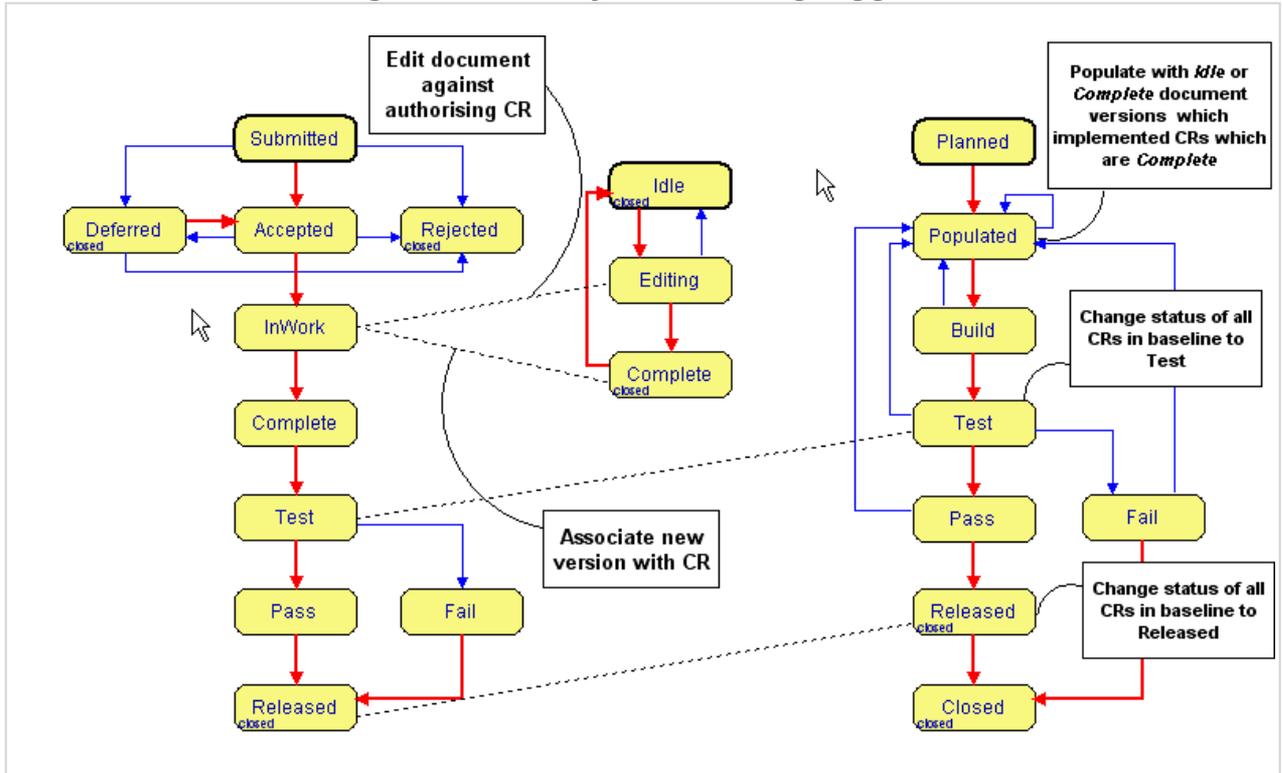
Life-cycles may also be used to enforce approval procedures and to trigger actions. For example, in the supplied configuration the `sw_release_cycle` implements the following:

1. The baseline is populated with the parts associated with selected CRs which are in a *Complete* state and targeted for the baseline.
2. When the baseline is progressed to *Build* then the parts in the baseline are checked out into a release workspace ready to be built for testing.
3. When the baseline is progressed to *Test* all the CRs associated with the baseline are also progressed to *Test*

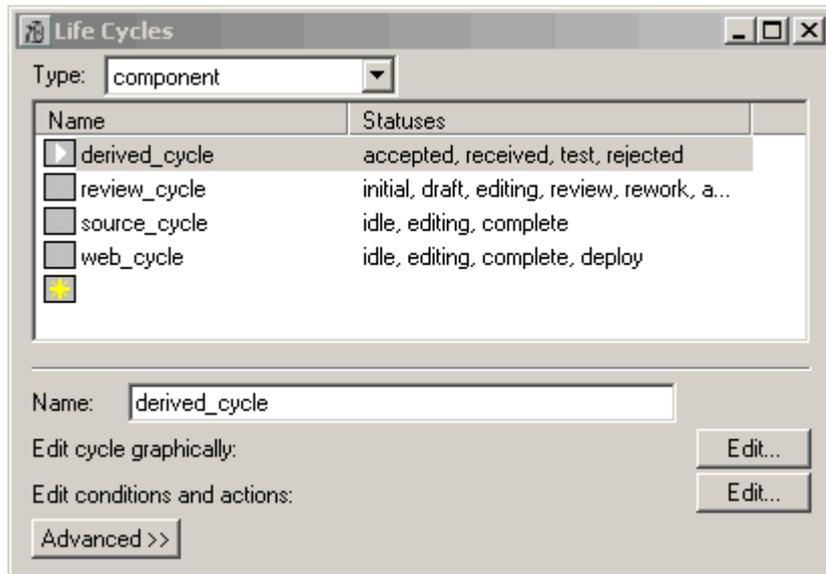
The above illustrates actions being used to trigger events.

Figure 8.2 illustrates the above process.

Figure 8.2: Life-cycles showing triggers



New cycles may be added and existing ones amended in ACCONFIG from **Plan | Life Cycles**.



The cycles are listed by **Type**, which determines the items with which the cycle is associated. Cycles may be of one of the following **Types**:

- component
- subsystem
- baseline
- cr

Cycles are added and managed by using the **Edit cycle graphically** button to access the [Cycle Editor](#). Clicking on the button, with a cycle selected, will open the Cycle Editor on the selected cycle.

The **Advanced** button provides access to alternative [non graphical cycle editing functions](#).

**Name** is an arbitrary name for a life-cycle (e.g. `source_cycle`, `document_cycle`); class definitions associate life-cycles with parts/ CRs/ baselines of a given class — see [Classes](#). The name must be comprised of alphanumeric characters and the `_` character and may not include a space character.

Each status has associated with it the following information:

- **Name** : The name of a status within the cycle (e.g. `Accepted`, `Released`). The name must be comprised of alphanumeric characters and the `_` character and may not include a space character.
- **Open or Closed** : Sets the *status type*, i.e. indicates whether an item in that status is considered to be "open" or "closed". It is up to the Administrator to determine how each status should be classified. The status type is used to classify items in that status in reports and visuals in ACE.
- **Create new version** : May only be used in life-cycles for components which have versions. Normally changing the status of a version changes that version's status (replacing the current status); if **Create new version** is selected a new version of the component will be created instead every time it enters this status. This facility *must* be used whenever the status actions cause a new version to be created (for check out statuses through an `issue -edit`); it *may* also be used against any other statuses if it is desired to retain a record of the version's previous status.
- **Attributes** : May be associated with a status which will affect the behaviour of items which are in that status. The attributes which may be specified vary depending on the cycle type as shown in the table below.

Attribute	Type	Description
CheckIn	component	Specifies that this status is to be used to check files in (i.e. return them from edit). The actions of this status should perform return from edit or equivalent <b>AllChange</b> command to implement the check in
CheckOut	component	Specifies that this status is to be used to check files out for edit. Statuses with this attribute should also have <b>Create new version</b> selected. The actions of this status should perform an issue for edit command or equivalent <b>AllChange</b> command to implement the check out
Accepted	cr	Specifies that this status is a status that indicates that a CR has been accepted for work, but no work has yet started. This is used for the CR Overview report
Auto Progress	cr, baseline	Specifies that this status will be progressed into automatically from an enclosing baseline life-cycle.
MSPProject Bline Start	cr	Specifies that the date of entry into this status should be taken as the anticipated (baseline) start date for the task on export to MS Project
MSPProject Finish	cr	Specifies that the date of entry into this status should be taken as the (actual) start date for the task on export to MS Project
MSPProject Start	cr	Specifies that the date of entry into this status should be taken as the (actual) finish date for the task on export to MS Project
Release	cr	Specifies that this status makes the CR a candidate for <i>release</i> . CRs which are in a <i>release</i> status are offered for selection by the <b>CRs for Baseline</b> function.
Release Testing	cr	Specifies that this status is a status that indicates a CR has been completed and is currently being tested for final release. This is used for the CR Overview report
Under Development	cr, baseline	Specifies that this status is a status that indicates that the work required to implement a CR or baseline is in progress. This is used for the CR Overview report and the <code>under_development</code> function available from the CR condition editor

Valid for Change	cr	Specifies that this status makes a CR valid for authorising a change to a component which is classified as <i>requiring a CR</i> , see <a href="#">Classes</a> . CRs in a <i>valid for change</i> status will be presented in the list of valid CRs on performing a Check Out/ issue for edit/ check out status change/ adding a new component/ checking in a new version without a preceding checkout for edit.
------------------	----	--

- **Status Entry/Exit Permissions:** Restricts the entry and exit of the currently selected status to roles which have been given a specific permission. Select from available roles. These may be defined using the cycle editor or in the [Command Access](#) window
- **Conditions and Actions:** Each status may also have Entry/Exit Conditions and Actions associated with it.

The Entry Condition is used to specify the condition(s) which must be satisfied in order to enter the status; Entry Action specifies the action(s) to be executed on entering the status. Exit Condition and Exit Action are used similarly when exiting the status.

Conditions may be used to enforce procedural requirements, for example, to enforce that particular CR field values are entered prior to entering a certain state of the life-cycle (e.g. priority must be allocated before a CR is accepted).

Actions may be used, for example, to trigger other **AllChange** actions (such as checking out a part) or invoke external tools such as mailing tools. Actions must be used to implement the CheckIn/Out actions associated with the corresponding status attributes.

By using conditions and actions procedures may be highly automated.

Certain predefined Conditions and Actions may be applied to a status using the [Cycle Editor](#), other arbitrary site specific conditions and actions may be written in ACCEL by editing the cycles.ac file. Conditions and actions may also be written directly in ACCEL and may be modified using the **Edit** button in the edit conditions and action section of the window.

Predefined Conditions which may be specified include:

Condition	Type	Description
<b>User in CR Field</b>	CR	Enforces that the current user is named in a specified field of the CR. Has one parameter, <i>Field name</i> , the name of a field containing the user name.
<b>Must Have CRs Affected</b>	CR	Enforces that the CR has CRs affected of a specified class. Has one parameter, <i>Class</i> , specifying one or more classes of items which should appear as CRs affected.
<b>OK to Complete CR</b>	CR	Enforces that the CR must be in a state in which it can be completed. This means that each part affected must have a version solved and there must be no parts checked out for edit against this CR
<b>CRs Affected In Status</b>	CR	Enforces that the CRs affected are in one of the statuses specified. Has 2 parameters, <i>Class</i> , the class of the CRs affected to check, and <i>Statuses</i> , specifying that the CRs affected must be in one of these statuses. Special statuses ( <i>open</i> ) and ( <i>closed</i> ) specify that the status must be an open or a closed status respectively.
<b>Field Must Have Value</b>	Any	Enforces that the specified field has a value. Has 1 parameter, <i>Field</i> , the field for which a value must be specified.

<b>Field Must Have Specified Value</b>	Any	<p>Enforces that the specified field has a specified value. Has 3 parameters, <i>Field</i>, the field for which a value must be specified, <i>Operator</i>, the comparison operator to be used and <i>Value</i>, the value to which the <i>Operator</i> is applied. The <i>Operators</i> offered will vary according to the type of the <i>Field</i> and the <i>Values</i> offered and the means of selecting the values, will depend on the values in the field's definition. Note though that fields whose value lists are defined as ACCEL expressions do not offer the values as the ACCEL cannot be evaluated at that time. Fields with 'known' types, e.g. user fields, will offer an appropriate list of values where possible.</p> <p>Values may be defined as ACCEL expressions by prefixing them with an ampersand (&amp;) character. 'Truth' fields will, as per the condition editor, offer operators of "Yes", "No" and "Empty", with no value selectable.</p>
<b>Fields Must Have Value</b>	Any	<p>Enforces that the specified fields all have a value. Has 1 parameter, <i>Fields</i>, the fields for which a value must be specified.</p> <p>This enables just one error message to be issued for testing that multiple fields have a value compared with multiple Field Must Have Value conditions each with an error.</p>
<b>Text Section must have value</b>	CR	<p>Enforces that the CR text section specified has a value - i.e. some text has been entered</p>

Predefined Actions which may be specified include:

<b>Action</b>	<b>Type</b>	<b>Description</b>
<b>Mail (Dynamic)</b>	Any	<p>Sends mail to a user named in a field of the current item. Has 4 parameters, <i>Fieldname</i>, the name of a field containing the name of the user to send mail to, <i>Subject</i>, the subject of the email (may be an ACCEL expression), <i>Text</i>, the text of the email to send (may be an ACCEL expression), and <i>Send to Self</i>: whether to send the mail if the named user is the current user.</p> <p><i>Subject</i> and <i>Text</i> may be plain text, or if starting with &amp; may be an ACCEL expression. To include a URL link to a specified item in the email use <a href="#">GenerateURL</a> ACCEL function. See also <a href="#">Integration with Mail</a></p>
<b>Mail (Static)</b>	Any	<p>Sends mail to a specified user. Has 4 parameters, <i>User</i>, the name of the user to send the mail to, <i>Subject</i>, the subject of the email (may be an ACCEL expression), <i>Text</i>, the text of the email (may be an ACCEL expression), and <i>Send to Self</i>, whether to send the mail if the named user is the current user.</p> <p><i>Subject</i> and <i>Text</i> may be plain text, or if starting with &amp; may be an ACCEL expression. To include a URL link to a specified item in the email use <a href="#">GenerateURL</a> ACCEL function. See also <a href="#">Integration with Mail</a></p>
<b>Mail Role</b>	Any	<p>Sends mail to all users with the specified role. Has 5 parameters, <i>Role</i>, the name of the role the members of which are to be mailed, <i>Top Part</i>, the part for which the role must apply, <i>Subject</i>, the subject of the email (may be an ACCEL expression), <i>Text</i>, the text of the email to send (may be an ACCEL expression), and <i>Send to Self</i>: whether to send the mail if the named user is the current user.</p> <p><i>Subject</i> and <i>Text</i> may be plain text, or if starting with &amp; may be</p>

		an ACCEL expression. To include a URL link to a specified item in the email use <a href="#">GenerateURL</a> ACCEL function. See also <a href="#">Integration with Mail</a>
<b>Lock CR</b>	CR	Locks the CR if there is a Locked field defined
<b>Lock Baseline</b>	Baseline	Locks the Baseline
<b>Unlock CR</b>	CR	Unlocks the CR if there is a Locked field defined
<b>Unlock Baseline</b>	Baseline	Unlocks the Baseline
<b>Check In New Version</b>	Component	Checks in a new version of the component which was previously not checked out for edit
<b>Check Out Read-only</b>	Component	Checks Out read only the part to the current workspace.
<b>Check In Read-only</b>	Component	Checks In from Read Only the part from the current workspace
<b>Check In Edit</b>	Component	Checks In from Edit the part from the current workspace
<b>Check In Discard</b>	Component	Checks In discarding changes the part from the current workspace
<b>Check Out Edit</b>	Component	Checks out the part for edit to the current workspace
<b>Increment Revision</b>	Component	<p>Increments a field which has the <a href="#">Revision Number attribute</a>. Has 4 parameters:</p> <ul style="list-style-type: none"> <li>○ <i>Fieldname</i>: the name of the Revision Number field</li> <li>○ <i>Increment</i>: a space separated list of the Revision Counters to increment</li> <li>○ <i>Reset</i>: a space separated list of the Revision Counters to reset to their initial value</li> <li>○ <i>Blank</i>: : a space separated list of the Revision Counters to blank (i.e. set to no value)</li> </ul>
<b>Auto-progress</b>	Any	Progresses the cycle on to another status automatically. Has one parameter, <i>To Status</i> , the status to auto-progress to.
<b>Check Out Edit Parts Affected</b>	CR	Checks out the parts affected by the CR for edit to the current workspace.
<b>Assign CR</b>	CR	Assigns the CR to a user/group which will be prompted for.
<b>Assign CR To Field</b>	CR	Assigns the CR to the user in the specified field. Has one parameter, <i>Fieldname</i> , the name of a field containing the user name to whom the CR should be assigned.
<b>Assign CR To Role</b>	CR	<p>Assigns the CR to a user who has a specified role. Has three parameters, <i>Top Part</i>, the top part to be used for determining who has the role, <i>Role</i>, which the user must have (for the top-part), and <i>CR Number</i>, the number of the CR to assign to the role.</p> <p>If <i>CR Number</i> or <i>Top Part</i> are not specified (or empty) then the current CR number and content of the current CR TopPart field will be used</p>
<b>Assign CR (Expression)</b>	CR	Assigns the CR to the user which is returned as the result of an ACCEL expression
<b>Progress all CRs</b>	Baseline	Changes the status of all CRs solved by a baseline which are in a specified status to a new status
<b>CRs For Baseline</b>	Baseline	Populate the baseline with the predecessor baseline (if any) plus the changes required for specified CRs. See <a href="#">Baseline Release Management</a> for an example of this.
<b>Check Out Baseline</b>	Baseline	Checks out the baseline to a specified workspace

<b>Check Out Baseline (Dynamic)</b>	Baseline	Checks out the baseline to a workspace named in a specified field
<b>Populate from Predecessor</b>	Baseline	Copies the content of the predecessor baseline if specified in a field named <b>Pred Baseline</b> (allowing for nomenclature mapping of baseline) to this baseline, clearing out any existing contents of this baseline first

If no condition is specified then it is always permissible to enter/exit the status providing you have the right permission; if no action is specified none is performed.

- **Votes:** Allows decisions to be made as to what progression should be made based on a consensus opinion at a status in the life-cycle. These may be defined using the cycle editor or in the [Vote Definitions](#) window
- **Vote Permissions:** Allows the permissions to ignore a vote and for passing to the next voters in a serial vote to be specified.. Select from available roles. These may be defined using the cycle editor or in the [Command Access](#) window

If **Options | Check ACCEL Syntax** is selected, then on return from ACEDIT to ACCONFIG the syntax of the ACCEL code in the life-cycles will be checked. If there is a syntax error then you will be offered the option to re- edit the file or discard your changes and revert to the last syntactically correct file.

The conditions and actions for each status in a life-cycle are specified according to the following format:

```

entrycond
  ACCEL code for condition
end
entryaction
  ACCEL code for action
end
exitcond
  ACCEL code for condition
end
exitaction
  ACCEL code for action
end

```

Code inserted by the Cycle Editor for predefined conditions and actions is identified by special comments of the form:

```

### ACCONFIG_INSERT(<condition/action name>)
call(<condition/action_function>, {<parameter>,});
### END_ACCONFIG_INSERT

```

For example for the **Field must have value** condition might look like:

```

### ACCONFIG_INSERT(FieldMustHaveValue)
call(FieldMustHaveValue, "pa_Comment", "");
### END_ACCONFIG_INSERT

```

These comments, and the code between them, should not be edited directly.

In addition to the above another life-cycle option, [ReviewCycles](#), may specify life-cycles which have a review type of cycle and which contain a 'draft' status. Review Cycles are those based around a review process with the document reviewer defined on an arbitrary field called Reviewer, and check-in statuses of 'Draft' and 'Approved'. Both are valid progressions from the initial status of the life-cycle as shown in the [review cycle](#) in the standard template.

On entry to the initial status of the life-cycle, if the user is the Reviewer and there is more than one possible progression they are offered a choice of progression from the initial status. This allows the user to specify whether the first version of the part is *Draft* or *Approved*. If the user is not the Reviewer, then the status (as

specified in [ReviewCycles](#) option) will be progressed to *Draft* since only the reviewer may approve a version.

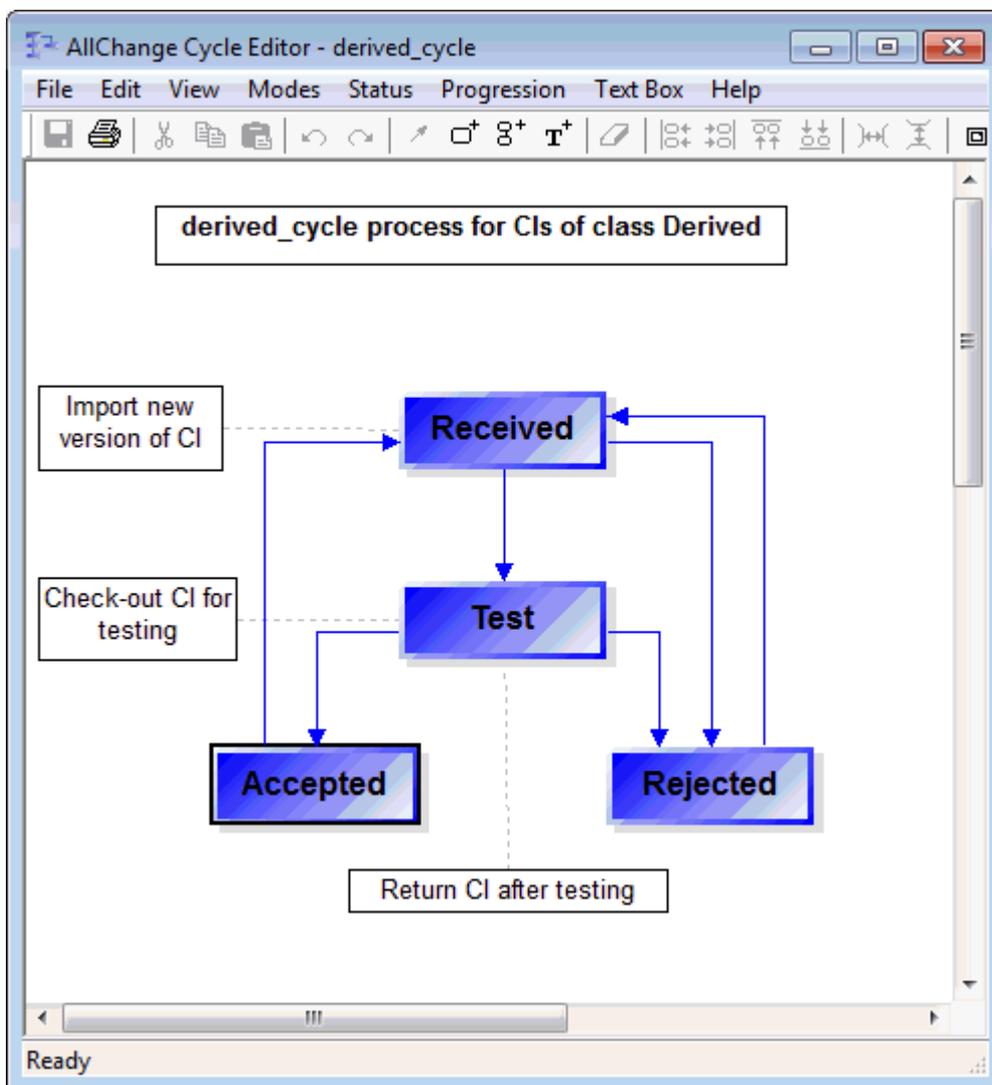
For details of the life-cycles defined in the supplied project templates see:

- [Intasoft Standard Cycles](#)
- [ITIL Cycles](#)
- [IntaChange Integrate Cycles](#)

The cycle definitions are stored in the `cycles.ac` configuration file in the project directory or system directory. The predefined conditions and actions available are defined in `predefs.acx` and the function definitions for these are in `cycfunc.ac`

### Cycle Editor

AllChange uses a Graphical Cycle Editor, allowing life-cycles to be created simply by drawing the statuses and linking them with progressions.



The Cycle Editor has four editing modes:

1. Select
2. Add Status

3. Add Progression
4. Add Text Box

The current mode may be selected from the toolbar, the 'Modes' menu, or from the right-click context menu of the diagram background. Pressing the **Escape** key while in 'Add Status', 'Add Progression' or 'Add Text Box' mode will cancel the current mode and revert to 'Select' mode.

Add Status, Add Progression and Add Text Box modes revert back Select mode as soon as the status, progression or text box has been added.

### Select Mode

Used to select and manipulate current objects (statuses, progressions and text boxes).

Whilst in this mode clicking on an object selects it, this is indicated by a dashed line.

Selected objects may be dragged to different positions in the diagram, and this may be fine tuned with pixel by pixel movement using the cursor keys.

To delete an object use the **Delete** key (or select **Delete** on the **Edit** menu)

#### Statuses

Statuses may be edited to change their open/closed status, attributes, access permissions and conditions and actions by double-clicking on a status, or by choosing **Edit Status** from the right-click menu.

The status menu controls order and appearance of the statuses within the cycle.

**Make Initial** sets the currently selected status to be the first step in the cycle.

The **Edit Status** dialog is available for selected statuses from the Edit or right-click menu and allows attributes of the status to be modified.

The **General** tab specifies the status Name, whether it is Open or Closed, whether a New Version is to be created and its **Attributes**. Note that statuses classified as *Closed* will have the word *closed* displayed in the bottom left of the status box.

The **Permissions** tab controls access to the status for selected roles within this project.

The **Predefined Conditions and Actions** tab allows the conditions for entry to and exit from the status to be specified. These are displayed as a tree of available conditions and actions which are applied to the status.

Initially there are 4 empty entries for **Entry Conditions**, **Entry Actions**, **Exit Conditions**, **Exit Actions**.

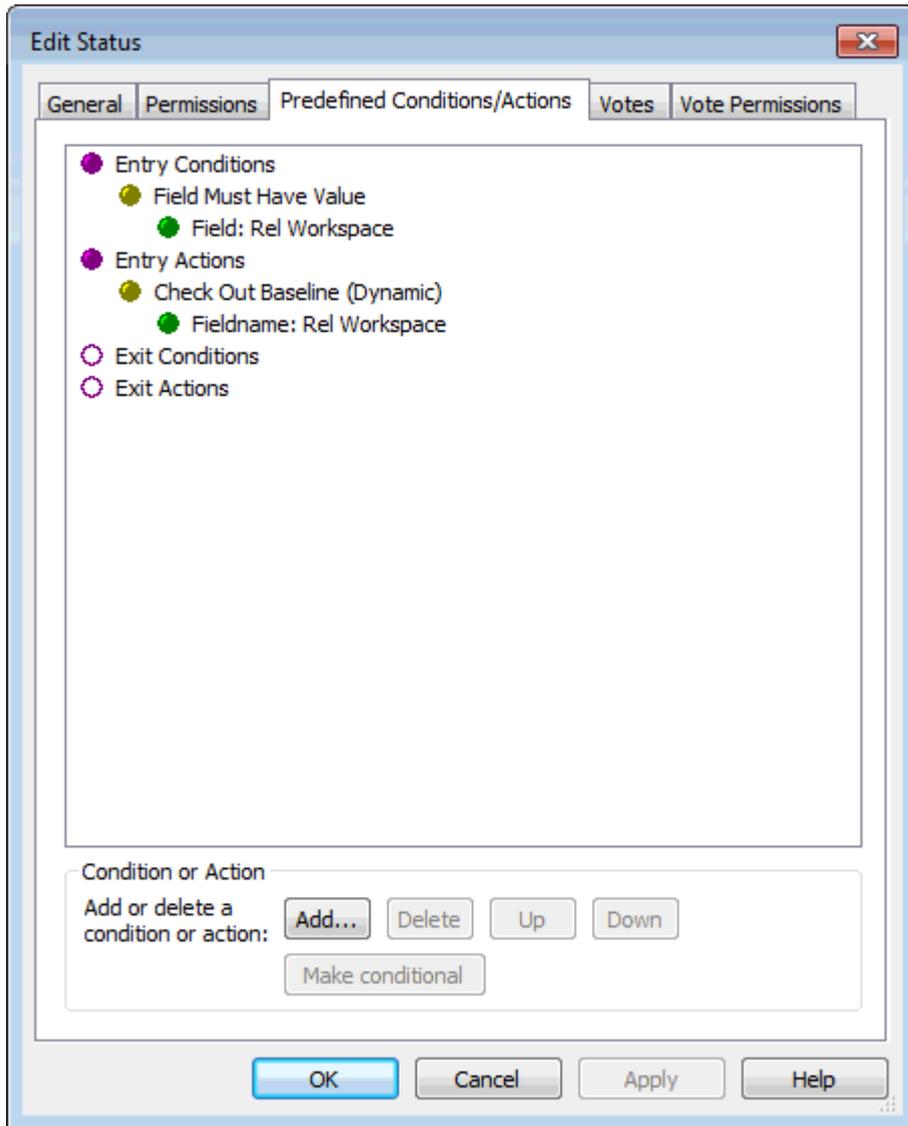
To add an entry to any of these select **Add** from the right click menu or the **Add** button. This will present a list of possible conditions or actions as appropriate. Once a condition/action has been selected it will be appended to the selected condition/action section and the item will expand it to show the required parameters.

These parameters may have their values set by means of drop-down lists of values, or edit fields where the value needs to be typed. Hovering the mouse cursor over the options or their parameters will show tips to assist with specifying the options and parameters. The conditions and actions available depend on the cycle's database type.

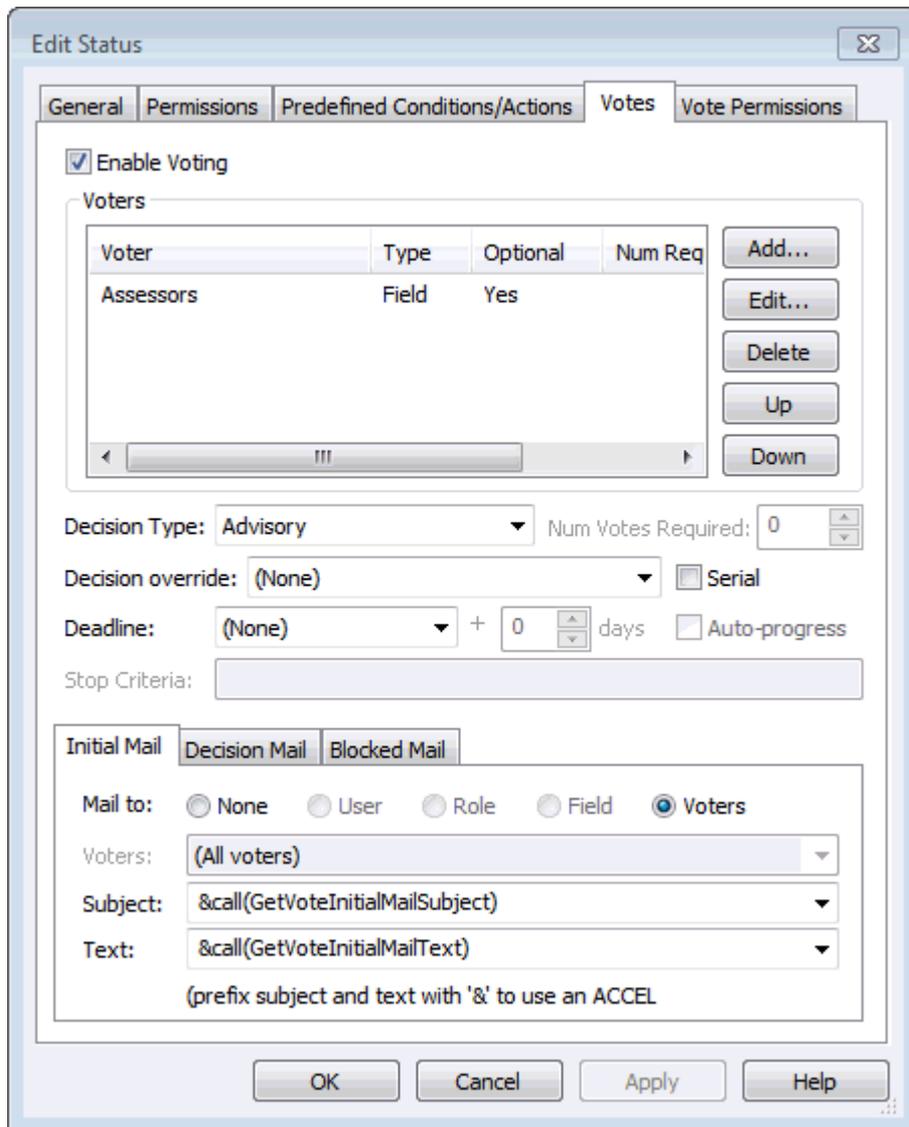
A condition or action may be made conditional, for example dependent on the status which is being progressed to (newstatus) or the status from which you have come (oldstatus). To make a condition or action conditional, select the **Make Conditional** button, or menu item on the context (right click) menu. This adds a condition field to the item. The condition will be in the form of an editable drop-down list, where a condition may be entered in ACCEL, or one of the preset conditions may be used. The preset conditions are dependent on whether the condition item is on the status's entry or exit. If on entry, presets will be added to test for the old status; if on exit, then the presets will be to test for the new status. The condition may be removed from the pre-defined action/condition by selecting the item and clicking the **Make unconditional** button or menu item on the context (right click) menu.

To delete a condition/action select it and select **Delete** from the right click menu or select the **Delete** button.

Adding a condition or action will cause ACCEL code to be inserted into the cycles definition file. Deleting a condition or action will automatically remove the corresponding ACCEL code from the cycles definition file.



The [Votes](#) tab allows a vote to be defined for the status. Select the **Enable Voting** to define a vote for the status.



The [Vote Permissions](#) tab allows vote specific permissions to be set for this status.

Multiple statuses may be selected by clicking on the background and dragging a selection box around the statuses required. When two or more statuses are selected, additional layout facilities become available from the Status menu or the right-click menu, allowing statuses to be aligned against the indicated section of the selected area.

**Align** options allow statuses to be aligned to left, right, top, or bottom and **Space Evenly** options allow the statuses to be spaced horizontally or vertically with the same distance apart.

#### Progressions

Clicking near a progression will show it selected with the segment nearest the cursor shown in grey along with any connections to statuses. A progression's connectors may be dragged to connect to a different status, or to a different point on the current status.

A progression is composed of individual **Segments** to which 'corners' or **Waypoints** and can be added using options from the progression menu. A selected progression appears as a dotted line, with the selected segment displayed in grey.

Progressions may have one of three styles (**Line Styles**): **Normal**, **Shortest** or **Orthogonal** (composed only of right angles).

- Normal progressions may feature diagonal sections and indirect routing around statuses but unlike Shortest or Orthogonal progressions 'Normal' progressions do not provide any automatic routing.
- Progressions with the 'Shortest' style will always be a single straight line, and will automatically re-route to take the shortest line between the two nearest connectors regardless of the position of other statuses.
- 'Orthogonal' progressions will have all of their segments either vertical or horizontal at all times and will be automatically routed as their end points or statuses are moved. Selected segments may be made orthogonal, that is, vertical or horizontal, which may then be dragged into position as required. To make the segments either side of a waypoint orthogonal, the waypoint can be dragged or clicked while holding the Ctrl key down.

**Line styles** can be changed for selected progressions from the right-click menu and the progression menu.

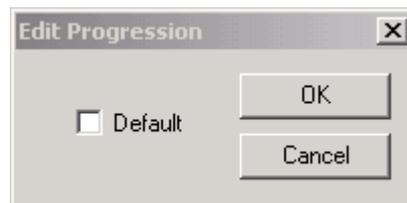
The progression menu controls alignment, routing and appearance of the progressions within the cycle.

**Make Default** specifies which progression is to be the default selected progression when the status of an item is changed and is available from the progression menu and the right-click menu.

Various **Segment** and **Waypoint** options on the progression menu allow individual sections of progressions to be moved or deleted.

**Follow Status Orbit** on the progression menu applies to circular progressions which loop back and return to the same status and will ensure the progression will follow the shortest possible route in an orbit around the status.

The **Edit Progressions** dialog is available for selected statuses from the Edit or right-click menu and allows a selected progression to be defined as the default selected progression when the status of an item is changed.



### Text Boxes

Text boxes may be edited to modify the textual content or display attributes such as alignment and font.

The **Text Box** menu is used to control the attributes of a text box and is available from the menu bar or the right click menu when a text box is selected.

Use **Edit Text** (or double click on the text box) to modify the text of the text box

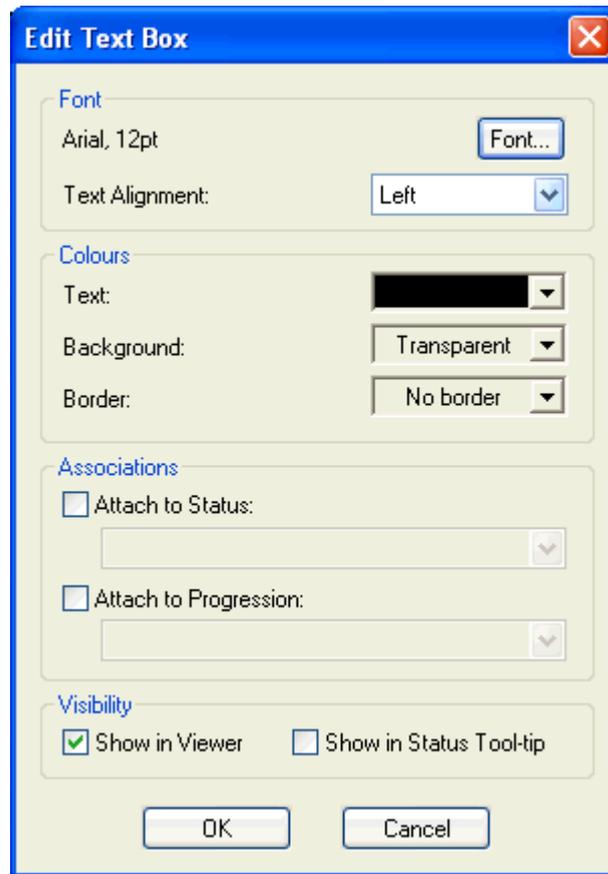
Use **Set Font** to change the font of the text in the text box

Use **Align Text** to change the alignment of the text within the text box

Use **Attach to Status** and **Attach to Progression** to attach the text-box to a status or progression - first select the status or progression then select the text box (using area selection or CTRL click) to be attached. Once attached to a status or progression it must be detached before it may be attached to a different object. *Note that although a text box may only be attached to a single status or progression, a status or progression may be attached to many text boxes.*

Use **Detach from Status** and **Detach from Progression** to detach the text-box from a status or progression.

All of the above may also be achieved using the **Edit Text Box** dialog which is available from **Properties** on the **Text Box** menu.



In the **Edit Text Box** dialog the text box may be attached to a named status or progression or the attached may be modified without first having to detach.

In addition the Edit Text Box dialog allows the visibility of the text box in the cycle viewer in ALLC HANGE to be controlled. Select **Show in Viewer** to allow the text box to be visible in the cycle viewer and select **Show in Status Tool-tip** or **Show in Progression Tool-tip** for the text in the text box to be added to the tool-tip for the attached status or progression in the cycle viewer.

The size of the text box may be modified by dragging the selection points as desired.

### Status Mode

Add Status mode is used to add statuses to the diagram. Whilst in this mode, a status is added on left click at the position of the mouse, automatically selecting the status name for edit in each case. The mode then reverts immediately to Select

### Progression Mode

Add progression mode is used to add progressions between statuses on the diagram. Progressions can be attached to any of the three connection points on the long sides of a status, or any of the two on the short sides.

Progressions are added by clicking on the *from* status, and then dragging to the *to* status. If a progression already exists between two statuses a warning dialog will appear.

Clicking on the same status twice applies acircular progression which will loop back and return to the same status.

Once the progression has been added the mode immediately reverts to Select.

## Text Box Mode

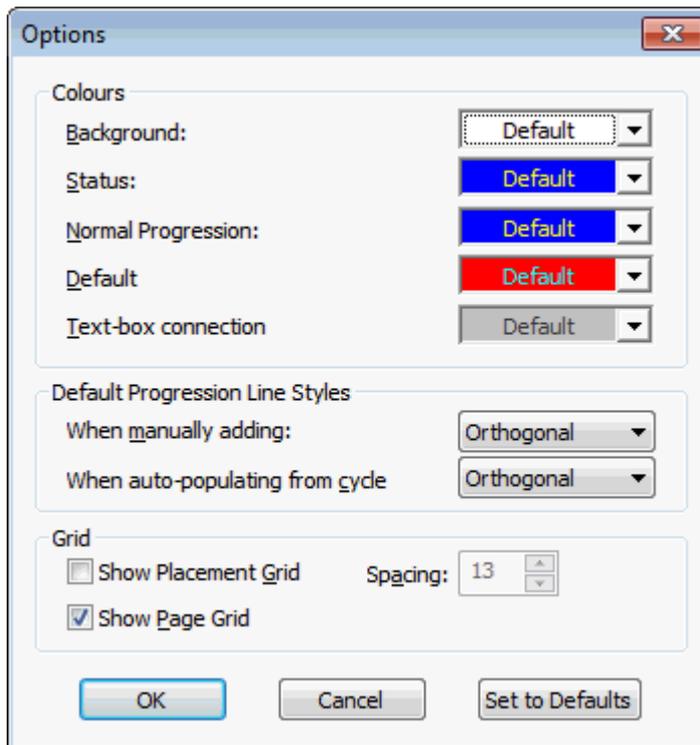
Text box mode is used to add text boxes to the life-cycle diagram. Text boxes may be used to annotate the life cycle and provide notes for administrators or guidance to end users.

Once the text box has been added the mode immediately reverts to Select.

## Display Options

The **View** menu allow various options to be selected which affect the display in the cycle editor.

- Using the **Zoom Menu** the layout view may be scaled as required including **Zoom to Fit** which will cause the life-cycle to be shown at a size to fit within the current cycle editor window.
- Using the **Options Menu** the colours used for statuses, progressions and the background within the cycle editor may be tailored to personal preferences. Grid settings may also be specified, whether the placement grid and page grid are shown together with the placement grid size may be tailored to personal settings. Default Progression Line Styles may also be specified for manually adding a progression and when automatically populated from the cycle.



Use **Set to Defaults** to revert to the default supplied settings

- **Grid** on the View Menu applies a grid background to the edit screen to help with spacing and alignment in the cycle. This is the same as the "Show Placement Grid" option on the Options dialog.

## Printing

Cycle diagrams may be printed using **Print** from the **File** menu. In addition to the standard print setup facilities, the **Print Setup** dialog has options to **Reduce large diagrams to fit the page** and **Enlarge small diagrams to fit the page**. These options will cause the life-cycle to be printed at the maximum size possible for the diagram to fill the page, Otherwise the cycle diagram will be printed at normal size and pagination will occur as required.

### Export to Image

The cycle diagram may be exported from the cycle editor as an image file for inclusion in word-processing documents etc. A variety of formats are supported which may be selected from the **Export to Image** dialog. The image file formats supported are Bitmap (BMP), Meta-file (WMF), JPG, TIF, GIF, PCX and PNG.

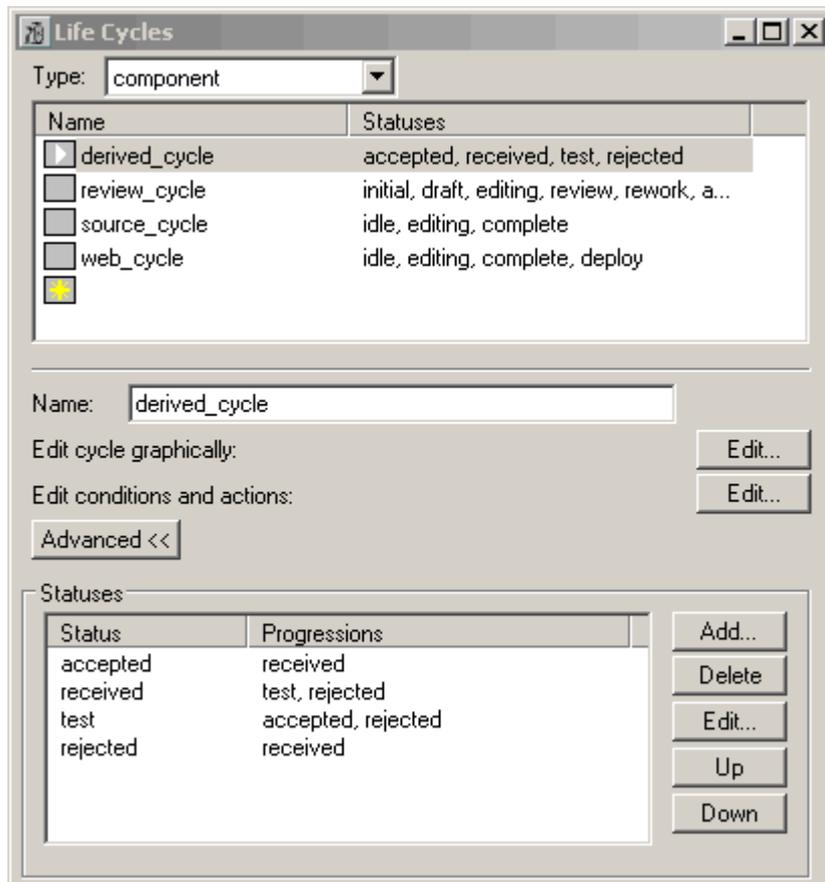
Some of the image file formats have settable options. The options appear on the **Export to Image** dialog. Meta-file (WMF), GIF, and PNG files can be made transparent, by checking the check-box on the dialog. JPG files can have their encoding set to Standard or Progressive, and can have their quality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

### Non-graphical Editing Options

The **Advanced >>** button expands the life-cycles window to show statuses for the currently selected cycle along with alternative, non-graphical editing options which duplicate the functions of the Cycle Editor. Note that changes to the cycle using the non graphical functions should be made with care as they may lead to an incompatibility with the cycle diagram when attempting to edit the cycle using the graphical functions (as new statuses and progressions will be auto drawn).

Clicking the button again will hide the controls and shrink the window.

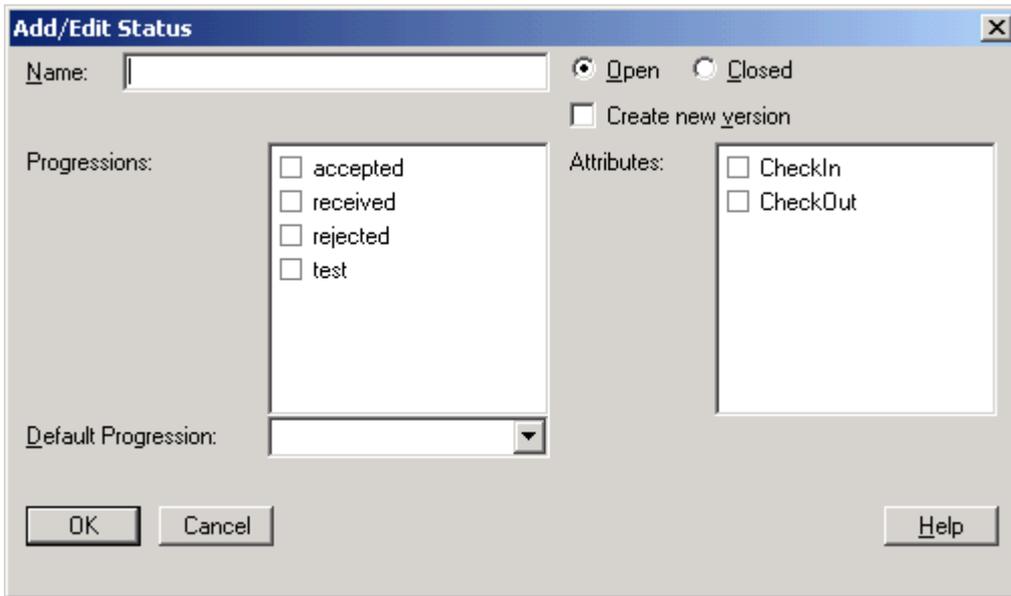


The Statuses list shows the statuses that exist in the cycle and the valid progressions.

The **Up** and **Down** buttons may be used to reorder the statuses. Although, except for the first status, the order is not significant it does aid readability. The first status will be used as the initial status when a new item is created which has the life-cycle.

**Delete** will delete the currently selected status.

**Add** allows a new status to be added and **Edit** allows the currently selected status to be altered.

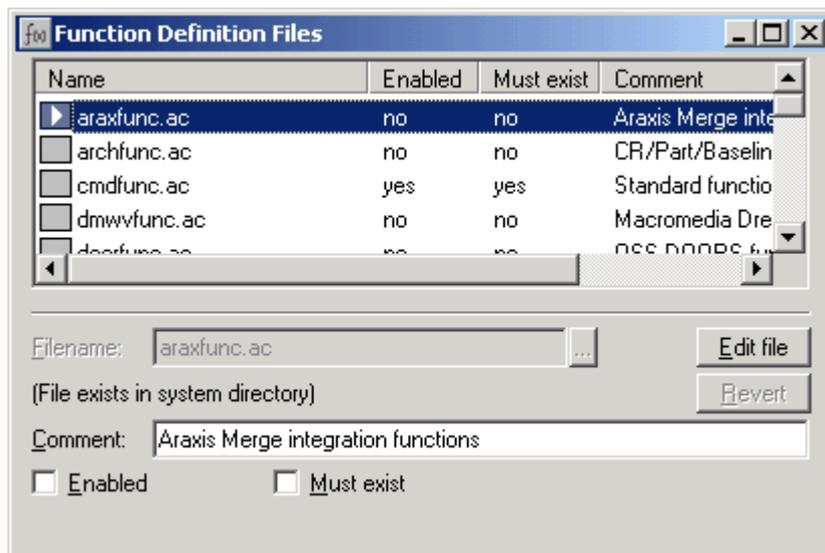


### Function Definition Files

#### About Function Definition Files

**AllChange** implements some of its functionality, and you may add your own functionality, by the use of functions written in ACCEL. These functions are stored in a series of ASCII text files which are supplied with **AllChange** together with site specific ones.

These files (and the functionality they implement) may be enabled/ disabled and the functions may be viewed/ modified using the **Function Files** option of the **Plan** menu in ACCONFIG.



The **Filename** is the name of a file containing ACCEL functions. This file will reside in the system directory if it was supplied with **AllChange** and has been unaltered. Any new or modified function definition files reside in the project directory.

The **Comment** will describe the purpose of the functions defined in the file.

If **Disabled** is selected then the functions implemented in that file may not be used from ACE. By default the following function files are disabled:

- Archiving (file `archfunc.ac`)
- Import from CSV File (file `csvfunc.ac`)
- Web browser/ HTML interface (file `htmlfunc.ac`)
- Araxis Interface (file `araxifunc.ac`)

**Must exist** specifies whether it is a fatal error if the file does not exist when ACE is invoked.

Under **Filename** a message will be displayed showing whether the file does exist and if so where it is located.

The **EditFile** button may be used to invoke the **AllChange** editor for the current file, see [The AllChange Editor](#) for details of using the editor. In order to continue to use `acconfig` whilst editing a function definition file (thus allowing the editing of multiple function files asynchronously), select **Options** | [Don't wait for Editor](#).

The configuration file in which this information is stored is `includes.ac` in the project or system directory.

## Command Functions

Various ACCEL functions are supplied with the standard **AllChange** configuration and used by the command definitions, the cycle definitions and provide various functions on ACE menus. These functions are supplied in the following function definition files:

- `cmdfunc.ac` for functions used by `commands.ac`
- `wspcfunc.ac` for workspace/pool functions
- `vcfunc.ac` for version control tool interface functions
- `utility.ac` for general useful functions
- `reportfunc.ac` for functions used in reports
- `getver.ac` for implementation of get version/baseline to directory
- `dragdropfunc.ac` for drag and drop functions
- `diffsfunc.ac` for diffs/merge functions
- `crsforblnline.ac` for implementation of Crs For Baseline
- `chkinoutfunc.ac` for checkin/out functions
- `acefunc.ac` for functions used by ace
- `csvfunc.ac` for functions for importing from csv files
- `archfunc.ac` for archiving functions

These also contain many useful utility functions which administrators may find useful when writing ACCEL code and configuring the system.

## Defining Your Own Functions

Site specific functions may be defined in additional files; `projfunc.ac` is predefined to allow project specific functions to be defined, others may be added as required.

User-defined functions are intended for writing high-level commands which the user may call from the **AllChange** interpreter or from ACE. They may also be used in the commands definition file itself, when writing conditions and actions, for conciseness or readability. See [User-defined Functions](#) for further details.

Normally functions are marked as read-only and so may not be redefined. However, administrative users (see [User Registration](#)) may redefine functions. This scheme means that end users can never redefine functions on whose behaviour the system may rely, but administrative users may use **Functions | Read Functions** or `read_functions()` to repeatedly re-read these files while they are being developed.

## Command Definitions

### About Command Definitions

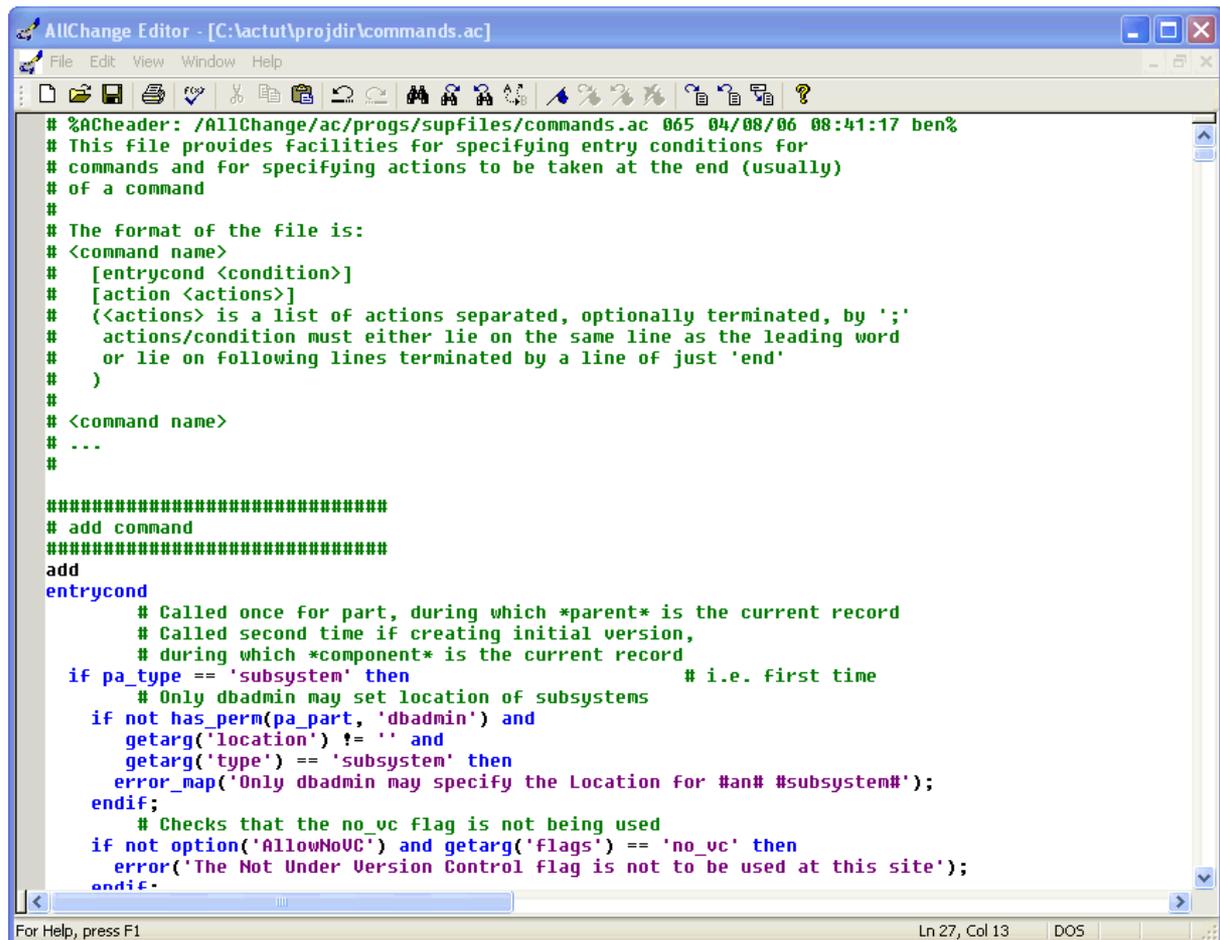
The commands definitions define the entry conditions and actions for nearly all **AllChange** commands (often referred to as the command definitions database).

In addition to AllChange commands, certain pseudo commands have entries in the command definitions. These include:

The command definitions are held as a text file called `commands.ac` which resides in the project directory or system directory. This is the key file to the functioning of most aspects of the **AllChange** system and may require tailoring to meet individual sites' requirements.

Command definitions may be amended by selecting **Plan | Command Definitions** in `ACCONFIG`.

This will invoke the **AllChange** text editor (ACEDIT) for the `commands.ac` file, see [The AllChange Editor](#) for details of using the editor. In order to continue to use `acconfig` whilst editing command definitions, select **Options | Don't wait for Editor**.



```

AllChange Editor - [C:\actut\projdir\commands.ac]
File Edit View Window Help
# %Acheader: /AllChange/ac/progs/supfiles/commands.ac 065 04/08/06 08:41:17 ben%
# This file provides facilities for specifying entry conditions for
# commands and for specifying actions to be taken at the end (usually)
# of a command
#
# The format of the file is:
# <command name>
# [entrycond <condition>]
# [action <actions>]
# (<actions> is a list of actions separated, optionally terminated, by ';'
#  actions/condition must either lie on the same line as the leading word
#  or lie on following lines terminated by a line of just 'end'
# )
#
# <command name>
# ---
#
#####
# add command
#####
add
entrycond
    # Called once for part, during which *parent* is the current record
    # Called second time if creating initial version,
    # during which *component* is the current record
    if pa_type == 'subsystem' then                # i.e. first time
        # Only dbadmin may set location of subsystems
        if not has_perm(pa_part, 'dbadmin') and
           getarg('location') != '' and
           getarg('type') == 'subsystem' then
            error_map('Only dbadmin may specify the Location for #an# #subsystem#');
        endif;
        # Checks that the no_vc flag is not being used
        if not option('AllowNoUC') and getarg('flags') == 'no_vc' then
            error('The Not Under Version Control flag is not to be used at this site');
        endif;
    endif;

```

Each command has a **Name** which gives the name of an **AllChange** command. There may be an entry for every **AllChange** command. There are also some special entries for pseudo commands including `interpret`, `monitor_action`, `readbaseline`, `readbaselinevote`, `readcr`, `readcrvote`, `readinstance`, `readmonitor`, `readpart`, `readpartvote`, `readbaselinestatuslog`, `readcrstatuslog`, `readpartstatuslog`, `readbaselinevote`, `readcrvote`, `readpartvote`, `voteinitiated`, `votedecision`, `voteblocked`.

Each command has associated with it an **Entry Condition** and an **Action**. These should consist of ACCELcode. If the **Options | Check ACCEL Syntax** option is selected, then any ACCEL expressions entered will have the syntax checked on exit from the editor.

Whenever the user executes an **AllChange** command the corresponding command entry condition is first evaluated and must succeed for the command to be allowed to proceed. After any internal processing the command actions are executed; should a fatal error occur during this stage the whole command will fail. By default actions are not executed either for parts which have the `no_file` flag set or for uses-type parts: this behaviour may be modified by the configuration options — see [Configuration Options](#).

If no condition is specified then no special conditions must be met in order to perform the command (the normal **Command Access** specifications still apply); if no action is specified no additional action is performed.

A sample setup for this file is supplied with the system; this may be customised as required by individual sites. Full details of the requirements for each command definition entry are given in [Command Definitions in Depth](#).

### File Format

This command definitions file is called `commands.ac`. The file consists of entries of the form:

```
command-name
    entrycond
    condition
end
    action
end
```

Each entry defines one command. Multiple entries are separated from each other by a single blank line; there should not be any other blank lines.

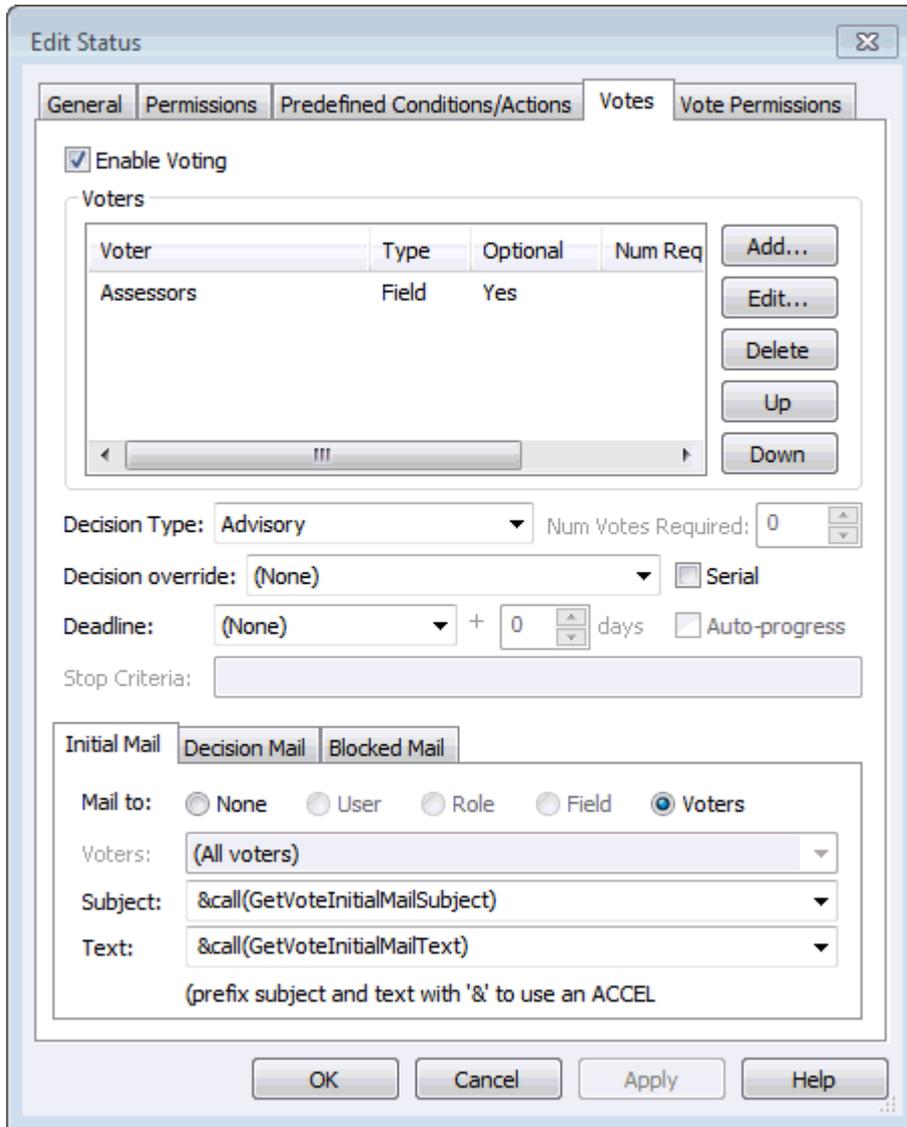
The *condition/ action* lines are optional.

### Vote Definitions

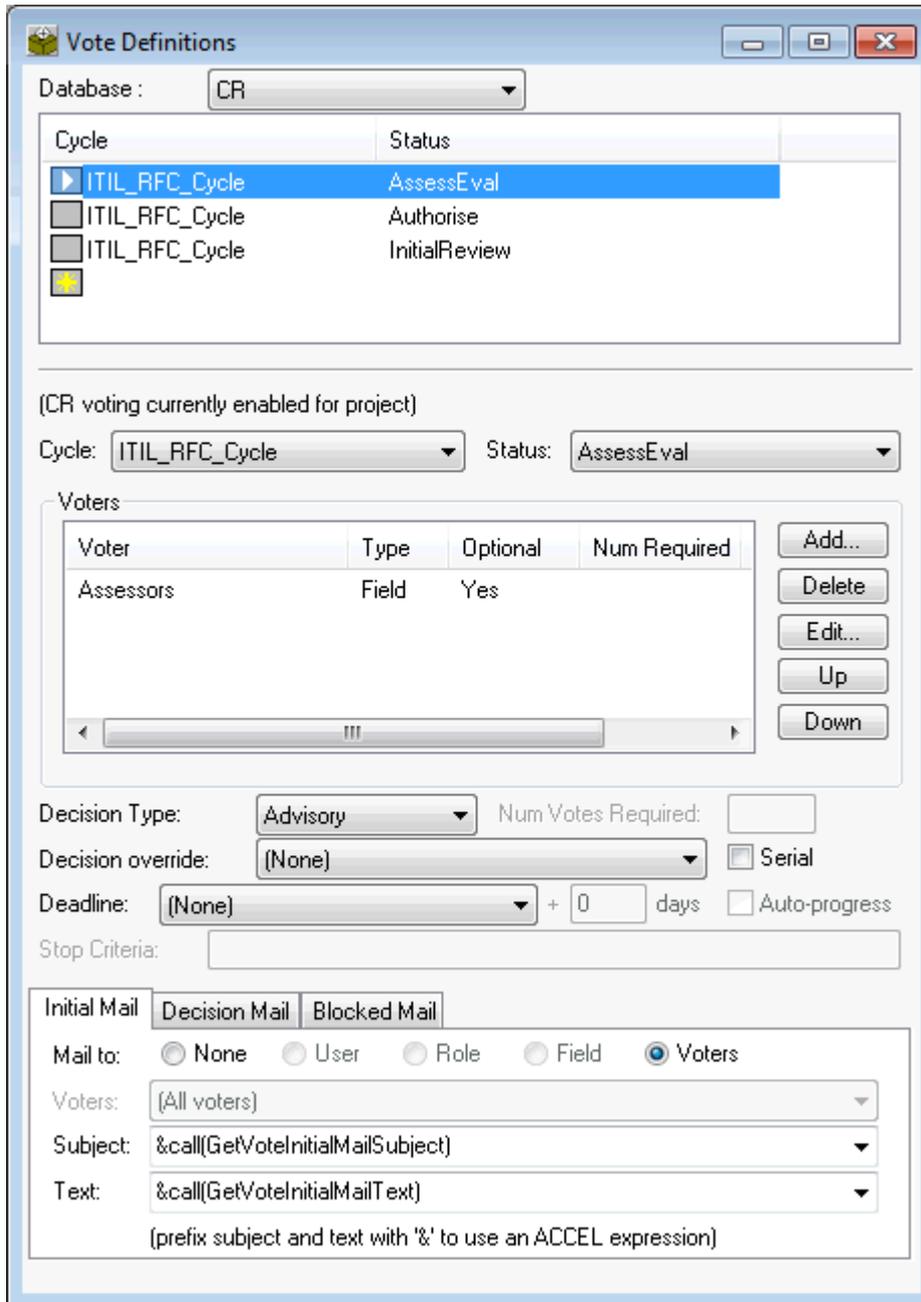
Voting allows decisions to be made based on a consensus opinion at defined points in a life-cycle. When a vote is in effect users can vote as to which status the item should be progressed to in their opinion. This may be used to enforce approval/review procedures.

Votes may be defined for parts, baselines and change requests. This may be enabled/disabled in the [Configuration Options](#).

Votes are defined either from within the **Edit Status** dialog in the [Cycle Editor](#),



or from the **Vote Definitions** window from the **Plan** menu.



Votes are defined for a given status of a given life-cycle. The possible outcomes of a vote are the statuses to which the cycle can be progressed from the vote status.

For each vote definition the following is specified:

**Voters:**

Any number of voters may be specified. Use the **Add**, **Delete** and **Edit** buttons to modify the list of voters. They may also be arranged using the **Up** and **Down** buttons although the order only has significance for Serial **Decision Type** votes.

The voters may be:

- Individual **AllChange** users
- A group
- A role
- A User/User List field

Each voter may be specified as **Optional** meaning that this voter does not have to cast a vote. If the **Voter Type** is a group, role or field, then the number of members of that group/role/field that are required to vote may be specified as well as the number allowed to vote

#### Decision Type

- **Advisory**: a decision is always reached, and all progressions are valid
- **Specified number**: allows a number to be specified as the number of votes required for a progression for a decision to be deemed to have been reached
- **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-thirds 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

#### Decision override

A status may be specified as a decision override status. If specified then one vote for the decision override status causes a decision to be reached regardless of the decision type of the vote definition.

#### Serial

Voters are asked to vote in turn in the order specified in the **Voters** list.

#### Deadline

A deadline may be specified. If a deadline is specified, then a decision is deemed to have been reached and no further votes may be cast nor may existing votes be altered. If no conclusive decision has been reached then the vote is blocked. Whether a decision may be reached depends on the **Decision Type**, and the votes cast.

The deadline may be specified as an offset from the vote start date (the date the status associated with the vote is entered), or from a date field for the item being voted on.

### Auto-progress

If Auto-progress is specified then as soon as a decision can be reached, the life-cycle is progressed to the decision status automatically. If the vote is blocked, then no status change is made. Auto-progress is not permitted for **Advisory Decision Type** votes nor for **Serial** votes.

### Stop Criteria

If the vote is a Serial vote, then Criteria may be specified to stop the vote, i.e. it is not passed on to any further voters. This is an ACCEL expression, which should return 'true' to stop the vote. This expression can be any ACCEL, and so can be as complex as needs be, allowing the exact criteria to be defined, for example, based on fields of the vote item or on votes cast.

### Initial Mail/Decision Mail/Blocked Mail

When a vote is started (i.e. when the status associated with the vote is entered) the **Initial Mail** is sent to the voters (just the first voters for a Serial vote) to inform them that they need to cast a vote if specified. The **Initial Mail** is also sent to the next voter(s) when the vote is passed on for a Serial vote.

Decision Mail is sent when a conclusive decision is reached. The users to mail may be specified as a specified user, a role or a user/user list field.

Blocked Mail is sent when a vote becomes blocked. The users to mail may be specified as a specified user, a role or a user/user list field.

The subject and text for each of these mails may be specified. This may be plain text or an ACCEL expression if prefixed with &. Default mail Subject and Text are supplied as ACCEL functions which may be modified if desired in `votefunc.ac`.

The vote definitions are stored in the configuration file `votedefs.acx` in the project directory.

Permission may be granted to allow a vote to be ignored and status progressed regardless by setting ignorevote permissions for the cycle/status and/or for voting in general, see [Command Access](#) and the [Cycle Editor](#).

Permissions may be specified for passing the vote to the next voters for a serial vote using `voteassnext` in [Command Access](#).

The pseudo-commands `voteinitiated`, `votedecision`, `voteblocked` may be used to tailor actions taken on these events occurring. By default these send out the appropriate email.

The nomenclature used to refer to the term vote and voter may be changed, see [Nomenclature Mapping](#).

### The Startup File

The startup file contains a set of **AllChange** commands which are to be invoked on startup before the user interacts with **AllChange**.

The startup file supplied in the *out-of-the-box* configuration, amongst other things, sets up some global ACCELvariables (e.g. the name of the default editor to be used, the mailing system to use), offers **AllChange** as a DDE and an OLE Server. It also executes any startup file found in the users home directory: by this means individual users of the system can have their own preferences obeyed on entry.

If any errors are encountered whilst processing the startup file, **AllChange** will quit (i.e. it will not start up).

The startup file comprises lines of **AllChange** commands expressed using the command line syntax. Details of the command syntax may be found in the **AllChange User Manual** command reference chapters.

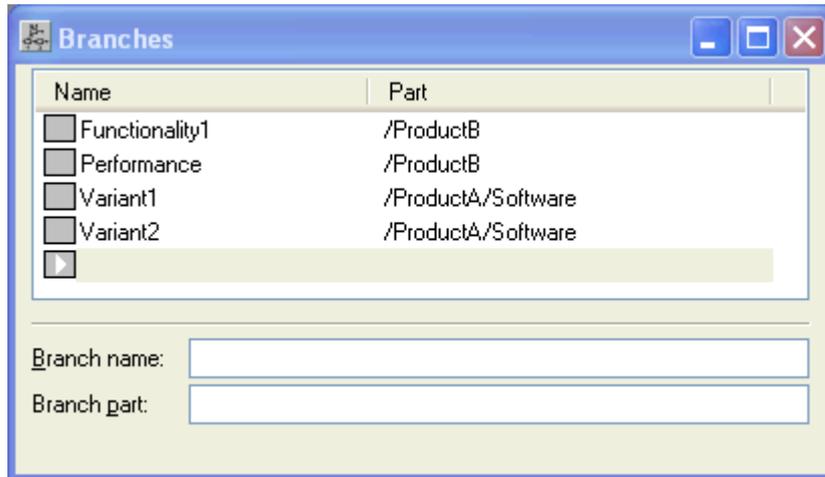
The startup file may be viewed/ modified from **ACCONFIGURING Plan | Startup File**. This will invoke ACEDIT for the startup file, see [The AllChange Editor](#) for details of using the editor.

The startup file is called `ac.ini`; it is sought first in the project directory, then in the system directory .

## Branch Names

Branches are used in **AllChange** to facilitate parallel development and variant management. Branches may be given names, these may be arbitrarily chosen by the user or chosen from a list of pre-defined names, whether arbitrary branch names may be used is specified in the [Configuration Options](#).

The list of pre-defined branch names may be managed from **Plan | Branch Names**.



Each branch name has the following information:

### Branch name:

This is the name for the branch.

### Branch part:

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

Branch name definitions may also be modified by users in ACE using Part | Edit Branches; the **Branch\_Add**, **Branch\_Edit** and **Branch\_Delete** function entries in [Command Access](#) may be used to control who may modify Branch Name definitions from within ACE.

The branch name definitions are stored in the configuration file `branches.ac` in the project directory.

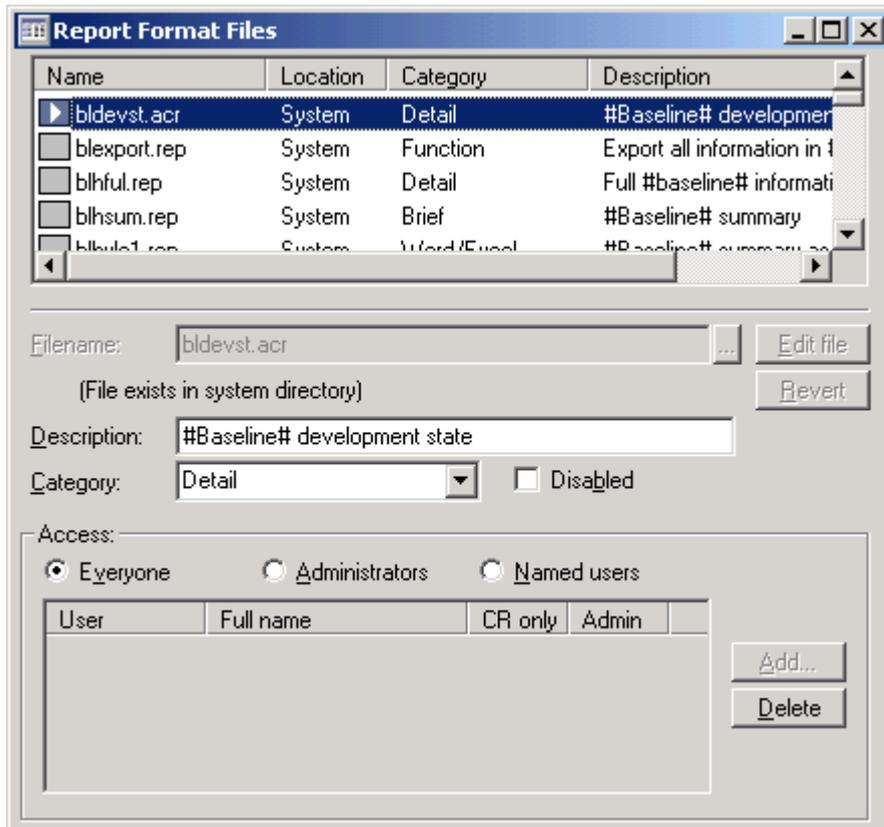
## Report Formats

Producing reports on the information stored as a result of your configuration management and change control activities using **AllChange** is an integral part of the configuration management process.

Many reports are supplied with the **AllChange** system and you may create your own and/or modify those supplied.

The content and layout of reports is controlled by *report format files* and these are listed in ACE in the **Report Wizard** and **Report Format** dialogs.

From within ACCONFIG the list of available report format files may be viewed and the reports may be modified from **Plan | Report Formats**.



If the currently selected file is a textual report format file (.rep) then it may be modified using the **Edit File** button. If this is a report format in the system directory, a copy will be made in the project directory and this is what is modified. The **Revert** button may be used to delete the format file from the project directory and hence *revert* to using the supplied one in the system directory.

ACREPORT format files may be modified using ACE, see Overview of ACREPORT for details.

Report format files have a .rep or .acr suffix and may be stored in the user's *home* directory for personal reports, the *project* directory for project specific reports and the *system* directory for the supplied reports.

The **Description** is displayed in the format lists shown from the Report dialogs and Report Wizard in ace.

The **Category** is used to determine which reports should be available for the various *wizard* report categories and which will be available in the Word report wizard.

The categories may be one of:

- Brief:** (Wiz) for summary information
- Detail:** (Wiz) for full detailed reports
- Function:** for reports used by supplied ACCEL functions
- Graph:** (Wiz) for reports producing graphical output for the supplied graph drawing tool
- HTML:** for reports which generate HTML format output
- Integration:** for reports used by the integrations with other applications such as the MCSCCI support, Explorer interface etc
- List:** for reports used in the **List** command
- Other:** for reports not to be shown in the *wizard* dialogs and not fitting any other category
- Status Log:** (Wiz) for reports on status logs
- Word Wizard:** for reports to be used from the **AllChange** Word Wizard
- Word/Excel:** (Wiz) for reports which export data directly to Word/ Excel

The categories marked with **Wiz** will be used by the report wizards, i.e. report format files of these categories will be presented in the wizards.

If **Disabled** is selected then the report format will not be available in ace.

Enabled report formats may have restricted availability if so desired by selecting **User Access** as:

- Everyone:** all users will have access
- Administrators:** only **AllChange** administrators will have access
- Named:** only the named users (or members of the named group(s)) will have access, in this case the required users should be added to the **Users** list

The configuration file in which this information is stored is report.sum in the *same* directory as the report format file, e.g. format files found in the project directory will be presented with the summary read in from the report.sum found in the project directory.

For details on report format files see [Creating and Modifying Report Formats](#).

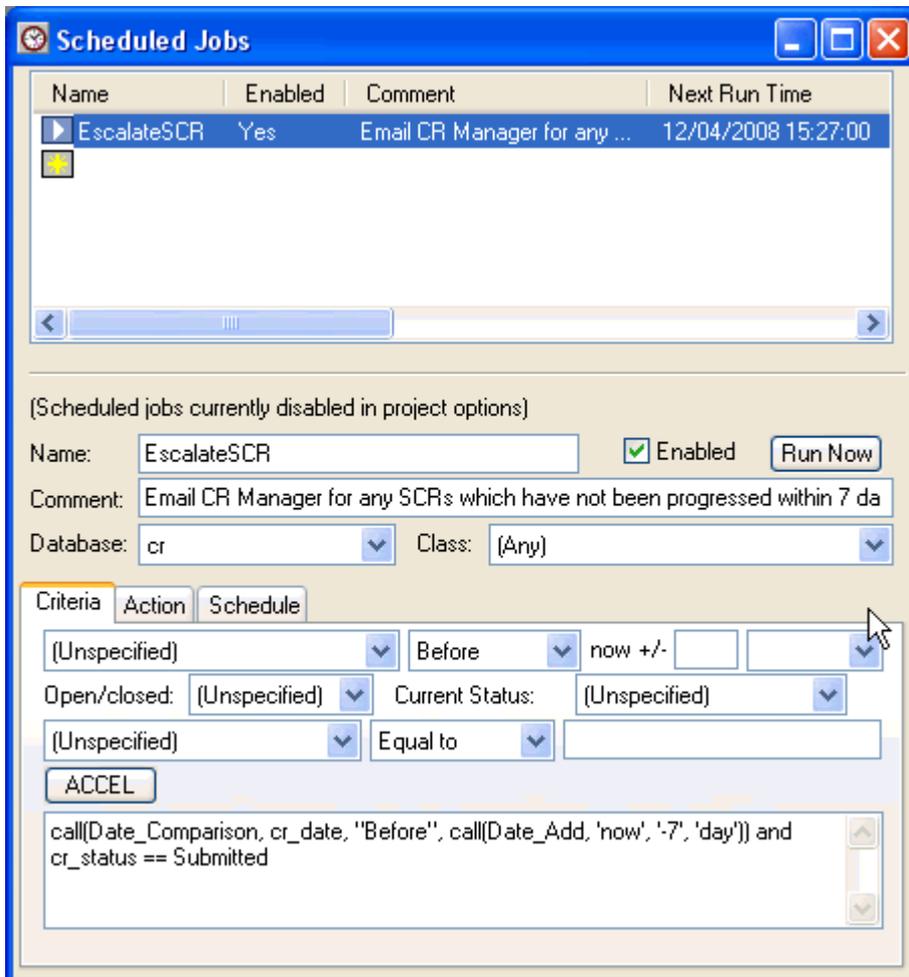
**Scheduled Jobs**

AllChange scheduled jobs allow activities to be scheduled to run at specified times and intervals. This may be used, for instance, to email users whose assigned CRs are overdue, escalate CRs if they have not been acted on promptly or run reports on daily activity.

Scheduled jobs must be enabled before any job definitions will have any effect - see [Enabling Scheduled Jobs](#).

Jobs are defined and edited in ACCONFIG from **Plan | Scheduled Jobs**.

The configuration file in which these settings are stored is `acjobs.acx` in the project directory



Each Job has the following information:

**Name**

The name the job is to be known as.

**Enabled**

If checked the job is enabled and will be run according to the specified schedule

**Comment**

A comment may be specified describing the job

**Database**

The database if any that the job is to be run on. If none specified then it is global.

**Class**

If a database is specified, a class may also be selected for which items within the database the job is applicable to.

**Criteria**

Criteria for running the job may be specified if the job is run against a Database.

**Action**

Specifies the action that is to be performed

**Schedule**

Specifies the times when the job is to be run.

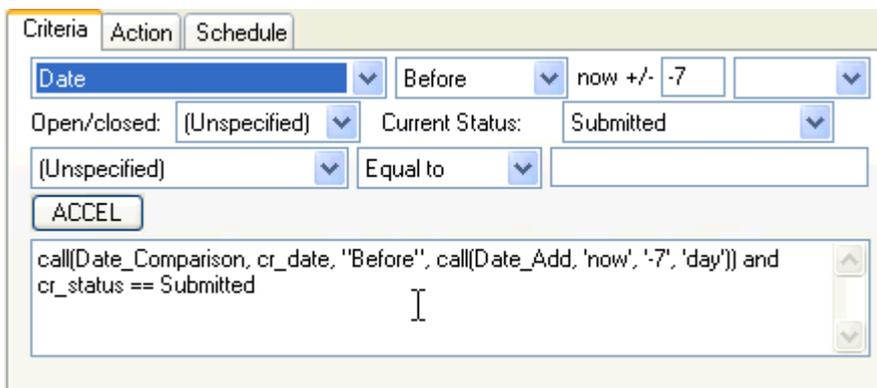
The **Run Now** button may be used to run the job now

*Criteria*

The criteria for running the job may be specified if the job is run against a database. For a job to run, the criteria must be satisfied for the specified database and class.

Criteria for a job consists of an ACCEL expression.

A simple expression builder may be used to assist with constructing the expression:



Any Date fields for the selected database and class may be selected together with a date based operator and a time of + or - a specified number of minutes, hours, days or weeks from *now* (the date and time at the point that the job is being run). This allows criteria such as:

- any CR which was created 7 days ago
- any CR whose Date Due is in 7 days time.

The expression builder also allows specification of whether the CR is in an Open or Closed status, and whether its Current Status is a specified status.

One additional field may be tested for specific values using the following operators:

- Equal to
- Not equal to

- Greater than
- Greater Equal
- Less than
- Less Equal

After setting the fields and values, clicking on the **ACCEL** button will generate the ACCEL.

However, any arbitrary ACCEL may be entered into the **ACCEL** edit control if a more complex expression than is supported by the expression builder is required.

### Action

The action for a job specifies what action is to be taken if the criteria are met.

The action may be specified as one of the following:

- **ACCEL Expression:** allows any arbitrary ACCEL expression to be evaluated.
- **Call User Function:** allows a named user-defined ACCEL function to be called. Any parameters to the function should be separated by a , (comma) and may be an ACCEL expression.
- **Email User:** allows an email to be sent to a user. The user to send the mail to may be:
  - an **AllChange** username,
  - a role - in which case any users having the role will be emailed
  - or a field defined using the 'User' or 'User List' data type, which will name the user or users to whom the email is sent.

The **Subject** and **Text** of the email may be specified as simple text or as an ACCEL expression by prefixing it with an ampersand character ('&'). To include a URL link to a specified item in the email use [GenerateURL](#) ACCEL function.

- **Check Open Votes:** checks all open votes to see whether they should be closed due to deadline expiry or other criteria
- **Assign CR:** allows a CR to be assigned to a specified user. This is only available for jobs on the CR database. The **User** to assign to may be:
  - an **AllChange** user or group
  - a field of User data type which will name the user to assign to
- **Change Status:** allows the status of the item matching the criteria to be changed to the specified status, providing the item's current status allows the progression. This is only available for jobs on a specified database
- **Change Field Value:** allows a specified field to be set to a given value. The new value may be specified as an ACCEL expression by prefixing with an ampersand ('&'). This is only available for jobs on a specified database
-

## Schedule

Jobs are scheduled to run according to the schedule settings.

The screenshot shows a 'Schedule' configuration window with the following details:

- Criteria** | **Action** | **Schedule** (selected tab)
- Schedule Job:** Daily (dropdown)
- Start Time:** 12:45 (input field) hh:mm (24 hour)
- Schedule Job Daily:** Every 1 day(s) (input field)
- Action Interval:** (Once) (dropdown)

The **Schedule** may be specified as one of: -

- **Every Minute:** the job will run every specified number of minutes
- **Hourly:** the job will run every specified number of hours at the time specified, and at the same minutes past each hour
- **Daily:** the job will run every specified number of days, at the specified time of the day
- **Weekly:** the job will run every specified number of weeks, on the selected days of the week, at the specified time.
- **Monthly:** the job will run on the specified day of each month at the specified time. If the day specified is later than the 28th day, then the job will only run in months that contain the specified day.
- **Once:** the job will run once only, at the specified date and time.

In addition, an **Action Interval** may be specified. This ensures that the action is not triggered any more frequently than the specified interval. The interval may be specified as a number of minutes, hours, days or weeks, or just once. For instance, it may be desirable to check for overdue CRs every hour, but for the action, e.g. email the assignee, to be carried out no more than once a day, in this case the schedule would be hourly, but the action interval would daily.

As each item that triggers the action needs to record when an action was triggered, the interval is only available when a database is specified. The action is recorded in the Status Log of the relevant item as a "Scheduled Job" entry, where the Arb1 field specifies the job name.

### Enabling Scheduled Jobs

In order to enable scheduled jobs for a project, the **Include this Project in Job Scheduling** option in the [Project Definition](#) must be set.

The jobs are run by a utility named ACJOBS (`acjobs.exe`) which is located in the **AllChange** executable directory.

In order to configure the scheduled jobs to run automatically, it is recommended that a new Windows Scheduled Task is set up on the server hosting the **AllChange** system, or another server which runs continually, to run ACJOBS automatically and at a suitable frequency.

To add ACJOBS to a server's Scheduled Tasks, you can use the **Add Scheduled Task Wizard** from the Windows **Control Panel**.

For Example, to set up the Task to run **AllChange** jobs every minute, between 9:00 and 18:00 every weekday, use the following steps:

- Run the **Add Scheduled Task wizard**
- In the page that prompts for the program to run, browse for `acjobs.exe` in the **AllChange** executable directory.
- In the next page, a name for the Task should be specified, for instance "AllChange Scheduled Jobs".
- Select **Daily** for **Perform this task**.
- After clicking **Next**, enter 09:00 for the **Start time**, and select **Weekdays**. Select a date from which you wish this to start
- On the next page enter the user name and password that the task should run as. Note that this is the **AllChange** user that jobs will run as and so must be a registered **AllChange** user. This user should also have the **dbsuperuser** role
- On the final page, select the "**Open advanced...**" option, and then click **Finish**.
- In the Properties dialog, select the **Schedule** tab, and click the **Advanced** button. In this dialog, select "Repeat Task", and specify **Every 1 minutes**. Select **Until Time**, and enter 18:00, then click **Ok**.
- Click **Ok** on the Properties dialog to save and schedule the Task.

Note that the most frequently a job may be scheduled in ACCONFIG is once per minute, in which case the Windows Scheduled Task should run ACJOBS once per minute. The Task may be scheduled less frequently if desired, but jobs defined to run more frequently will only be run - at most - at the lower frequency.

If ACJOBS is run with no arguments it will check each defined **AllChange** project in the system to see if job scheduling is enabled. If it is, then the utility will itself run ACJOBS with a command-line of:

```
acjobs.exe -project <project-name>
```

ACJOBS invoked with a command line argument of an **AllChange** project uses ACC to call an ACCELfunction named **RunScheduledJob** defined in `schedfunc.ac` with the job information. This causes the ACJOBS utility to read the defined jobs from the `acjobs.acx` file and run any jobs that are due at that time. The last run time, and newly calculated next run time is written to the `acjobtimes.acx` file in the project directory.

When a job is run, a temporary file is created in the project directory named after the job, in the form:

```
ac_<job-name>.acjob
```

this is deleted when the job is completed.

If the file exists when a job is due to start, the job is not started. Note that if for any reason the file is left behind by acjobs (for example, if the system crashes before the job completes), then the job will not run again until the temporary file is removed.

When running jobs, they are run as the **AllChange** user that invoked the ACJOBS. You should ensure that the user is set as a valid **AllChange** user. Also, the user should have the **dbsuperuser** role defined.

If errors occur while running scheduled jobs, either in ACJOBS or in the invoked ACC, the errors are written to the system's event log, and are viewable in the **Event Viewer**, available from Windows **Control Panel**.

## Defining Workspaces and Pools

### About Defining Workspaces and Pools

The workspaces and pools and other directory information required by **AllChange** may be defined from the **Workspaces** menu in ACCONFIG. In addition users may create and modify their own workspace definitions from within ACE from the **Workspace** menu.

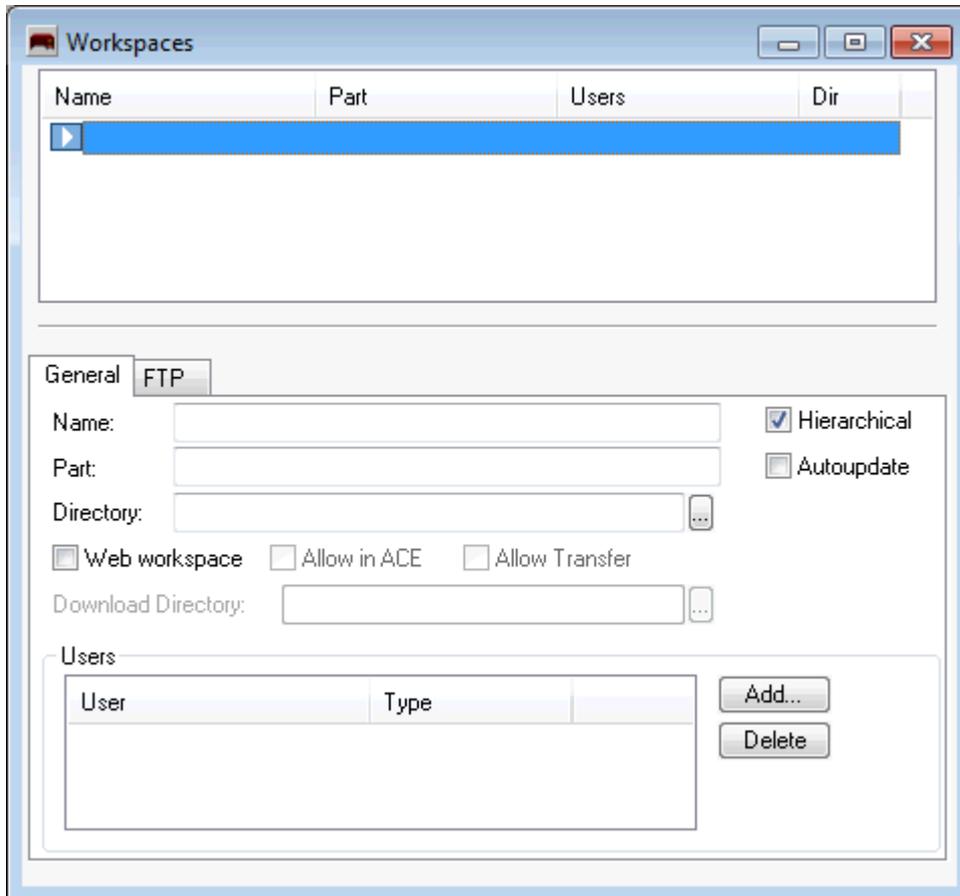
See the Checking files in and out of **AllChange** chapter of the **AllChange** User Manual for a discussion of the workspace and pool concepts.

## Workspaces

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.

New workspaces may be added and existing ones amended by using **Workspace | Workspaces** in ACCONFIG.

*Note that if workspaces are permitted to be defined from within ACE then this is incompatible with using ACCONFIG to define them. This is because ACCONFIG is not a multi-user application and any changes made to the workspace definitions could overwrite changes made at the same time from within ACE.*



Each workspace has the following **General** information:

### 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 partname e.g. `/product/software/source`. This may **not** be a user 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. If the directory specified does not exist ACCONFIG will offer to create it for you.

If the workspace is a web workspace it must have its root directory defined at a location which the

web server and **AllChange** server have access to. Permissions will need to be set up on this directory so that the user running as **AllChange** and the user running as the web server have read/write access.

### Hierarchical

If a workspace is hierarchical the physical workspace directory hierarchy must reflect that of the parts tree from the **Part** downward.

### Autoupdate

If this is selected then on attaching to the workspace **AllChange** will check to see if the workspace has the latest version of all the parts that are currently checked out to it for read only purposes. If not then the user will be offered to update to the latest versions.

### Web workspace

If a workspace is a web workspace then it may be used with the **AllChange** web browser interface. Only workspaces marked as web workspaces may be used with the web browser interface. The directory for a web workspace must be a directory that the web server can read/write to. The web server user and the **AllChange** user will both need read/write permissions for this directory.

### Allow in ACE

If selected for a web workspace then the workspace is allowed for use within **ACE** (the Windows interface to **AllChange**) as well as the browser interface.

### Allow transfer

If selected then users are permitted to upload and download files to/from their web workspace using the web browser interface to AllChange.

### Download Directory

If **Web workspace** is selected then a **Download Directory** may be specified. This is the default directory on the client machine for transferring files when doing a **Check Out**, **Download** or **Get Version to Directory**. This relates to the workspace directory, it mirrors the directory structure of the workspace directory. The user does have the option to change the directory from the default when downloading/transferring.

### 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** workspace for use when managing files on remote machines (e.g. Unix, VMS) and Web servers.

All the above **General** 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. [Figure 8.3](#) illustrates the movement of files when checked out and checked in to/ from an FTP workspace.

**Figure 8.3: FTP Workspaces**



FTP workspaces may be used for access to remote platforms from the Windows interface or for deployment to a web server.

In addition the following **FTP** information needs to be supplied (on the **FTP** tab of the **Workspace** window). **Note** that the configuration option **Enable FTP workspaces** must be enabled for the FTP workspace facilities to come into effect.

### FTP Workspace

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

**Host OS:** this should specify the host operating system/FTP server — see [Integrated FTP Support](#)

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

**Web deployment**

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

**Host OS:** this should specify the Web server operating system/FTP server — see [Integrated FTP Support](#)

**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.

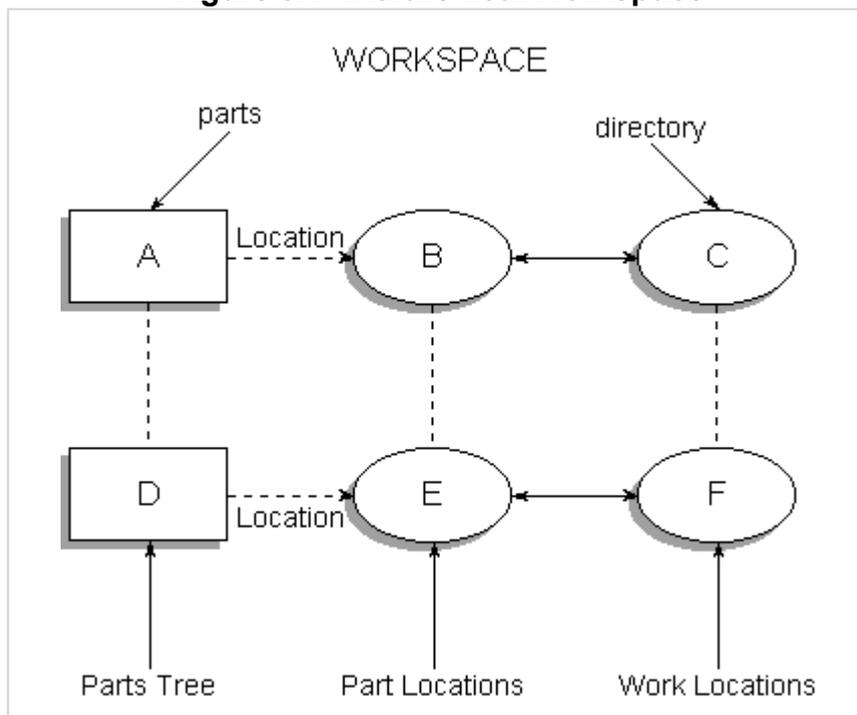
A workspace has a single associated part (which will be a subsystem ) called the *workspace part*. The intention is that all local work undertaken for descendants of this subsystem by users of the workspace will take place in the workspace. Depending on the system a workspace may be associated with the 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 is shared by multiple users, or there may be separate workspaces all associated with the same part for different users.

A workspace also has a single associated operating system directory called the *workspace directory*. Users of a workspace will require operating system permissions to alter files and subdirectories in this directory. For a simple (non-hierarchical) workspace all workfiles will reside in this directory.

Alternatively, a workspace can be defined as being *hierarchical*. In this case the directory structure of the workspace should reflect the directory structure corresponding to the parts tree (see Part Location ). Whenever the system needs to determine the location of a workfile for a part in a hierarchical workspace it computes a relative path from the location of the workspace part to the location of the specified part and appends this path to the workspace directory, i.e. the workfile is located relative to the workspace directory just as the part's location is located relative to the workspace part's location. (Should the location of the specified part not be relative to that of the workspace part the workfile will simply be placed in the workspace directory.)

[Figure 8.4](#) illustrates how the system relates the location of a workfile to the location of the corresponding part in a hierarchical workspace.

**Figure 8.4: Hierarchical Workspace**



As an example, suppose (under Windows) a hierarchical workspace were defined whose corresponding part was `/product (A)` and whose corresponding work directory was `c:\work (C)`. The location of `/product` is `k:\global\product (B)`. Issuing a version of `/product/ component` to this workspace would thus produce a workfile `c:\work\component`. If the location of `/product/source (D)` is `k:\global\product\source (E)` — which will be the case unless the part's location file is explicitly set otherwise — then the corresponding work directory will be `c:\work\source (F)`. Issuing a version of `/product/source/component2` would thus produce a workfile `c:\work\source\component2`. If the workspace were not hierarchical the work directory would always be `c:\work`.

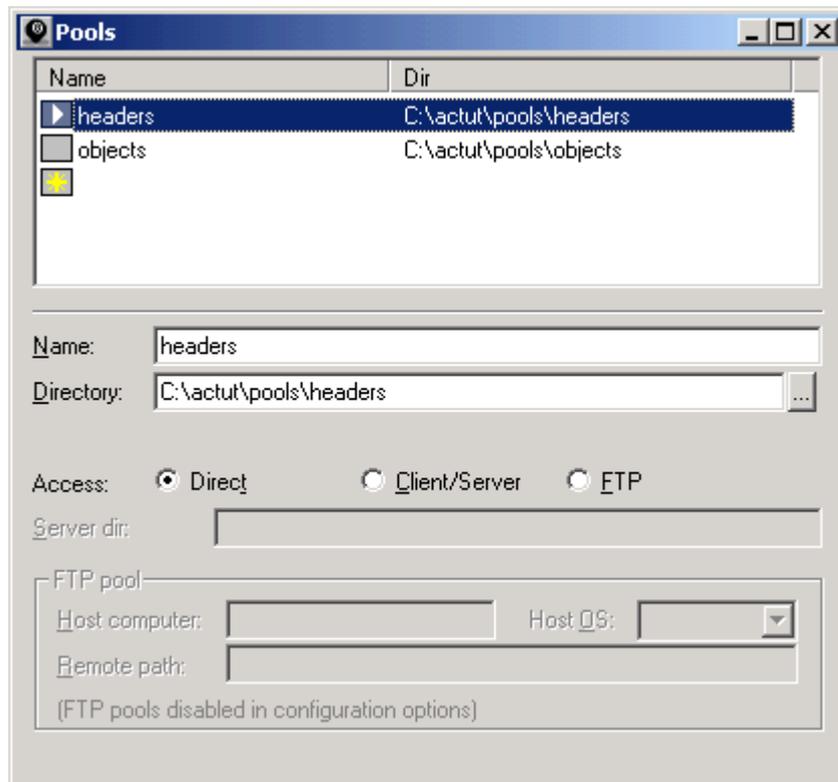
The workspace definitions are stored in the `workspcs.ac` and `ftpwspc.ac` configuration files in the project directory .

**Pools**

Pools are used for sharing objects for builds amongst different users. The objects held in pools are operating system files and the pools themselves are simply operating system directories.

Pools are closely connected to workspaces — see [Workspaces](#). Like workspaces, pools hold workfiles — files which may be examined with an editor, compiled, printed etc. — as opposed to the version history (VC) files which correspond to parts. 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 in an acceptable state for all users of the pool.

New pools may be added and existing ones amended by using **Workspace | Pools** option in ACCONFIG.



Each pool has the following information:

**Name**

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

**Directory**

Specifies the name of a physical directory into which files promoted to the pool will be placed. The

directory may be expressed as either a drive mapped directory or as a UNC. If the directory specified does not exist ACCONFIG will offer to create it for you.

**Access**

Specifies the method of accessing the pool directory when promoting files into the pool. **Access** may be one of:

- Direct:** this means that the user will have direct network access to the pool directory and the files may be placed in the pool by a straight forward copy of the file from the workspace directory.
- Client/Server:** this means that the files will be placed in the pool directory by means of client/server access to the pool directory.
- FTP:** this means that the *real* pool directory is a *remote* directory and the files will be placed in it by use of the **FTP** protocol. In this case the **Directory** is a local temporary location that the files pass through when going to/ from the remote directory. **Note** that the configuration option **Enable FTP pools** must be enabled for this mode of access to be performed.

**Server dir**

If **client/server** access is selected then the **Server dir** must be specified. This should be the directory path to the pool directory *from the server's point of view*.

**FTP pool**

If **FTP** access is selected then various information about the remote pool location must be specified:

- Host computer** should specify the name (or IP address) of the remote **Host** machine.
- Host OS** should specify the host computer operating system/FTP server — see [Integrated FTP Support](#)
- Remote path** should specify the path to the pool on the remote machine. This should be specified using the standard operating system path specification for the remote operating system.

Since objects are placed into pools from workspaces (see [Workspaces](#)), Pools and workspaces are closely associated. If the workspaces to be used with a pool are *hierarchical* then the pool must also reflect the same directory hierarchy.

Files should only be placed/ updated in pools by **Promoting** them from a **Workspace**. The files in the pool directory hierarchy may then be *read* (using an editor/ viewer) and *used* for build purposes. They should not however be updated in place in the pool directory.

When files are placed in a pool directory they are set to be *read-only* in order to protect them from accidental modification directly in the pool directory.

If **Direct** access to pools is used then users will require write access to the pool directory in order to promote files to the pool.

If **client/server** access is used then it is not necessary for users to have direct write access to the pool directory. Instead only the **AllChange** server requires write access and updates the pool on behalf of the user. Users will still require **Direct** read access to the pool directory in order to *use* the files in the pool. This further protects the pool directory from unauthorised modification. client/server access is only available if **AllChange** is being run in client/server mode of operation. If it is not then the pool will be accessed in **Direct** mode instead.

Local and FTP pools may be used with either local or FTP workspaces and for both FTP pools and FTP workspaces a transient local directory area is used to/ from which the FTP transfer occurs. [Figure 8.5](#) shows the movement of files between workspaces and pools in all the various combinations of FTP and non FTP.

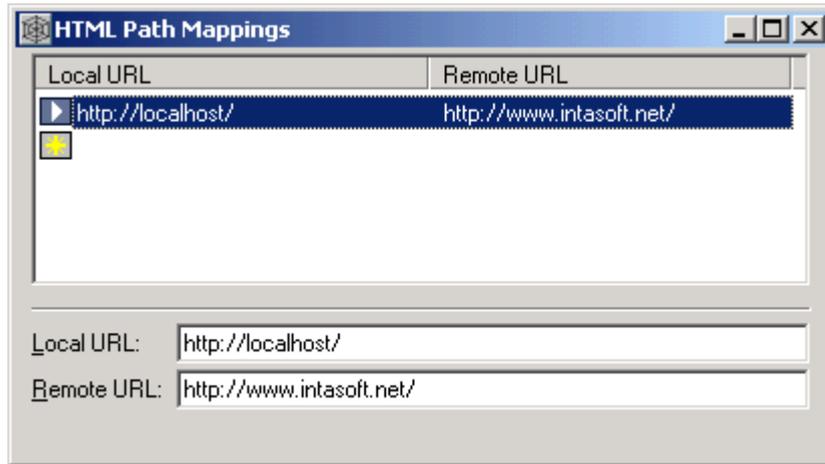
**Figure 8.5: Pools and Workspaces**



The pool definitions are stored in the `pools.ac` and `ftppool.ac` configuration files in the project directory.

### Web Mapping

This allows the definition of mappings between local and remote URLs for use with the Web deployment facilities within **AllChange**, see [Web Development Support](#).



The Web mappings are held in the `webmap.ac` configuration file in the project or system directory.

## Configuring the GUI

### About Configuring the GUI

Various aspects of the **AllChange** user interface (ACE) may be tailored to your specific requirements.

The facilities for doing this are available from the **GUI** menu in `ACCONFIG`.

The following may be tailored:

- Field Name Labels:** Both the labels and the type of arbitrary fields may be defined
- Menu Items:** What options are offered from the Main menu, the circumstances under which they are available and the actions performed may all be tailored
- Browsers:** The default browser column definitions and the columns available to users may be defined
- Condition Editor:** The fields and functions available from the condition editor may be tailored

### Field Name Definitions

A number of fields in parts, CRs, baselines, items affected, votes, instances and monitors databases are designated for holding "arbitrary" information. Such fields are intended to contain site-specific information. A policy should be established as to which fields, if any, are to be used and what they should contain.

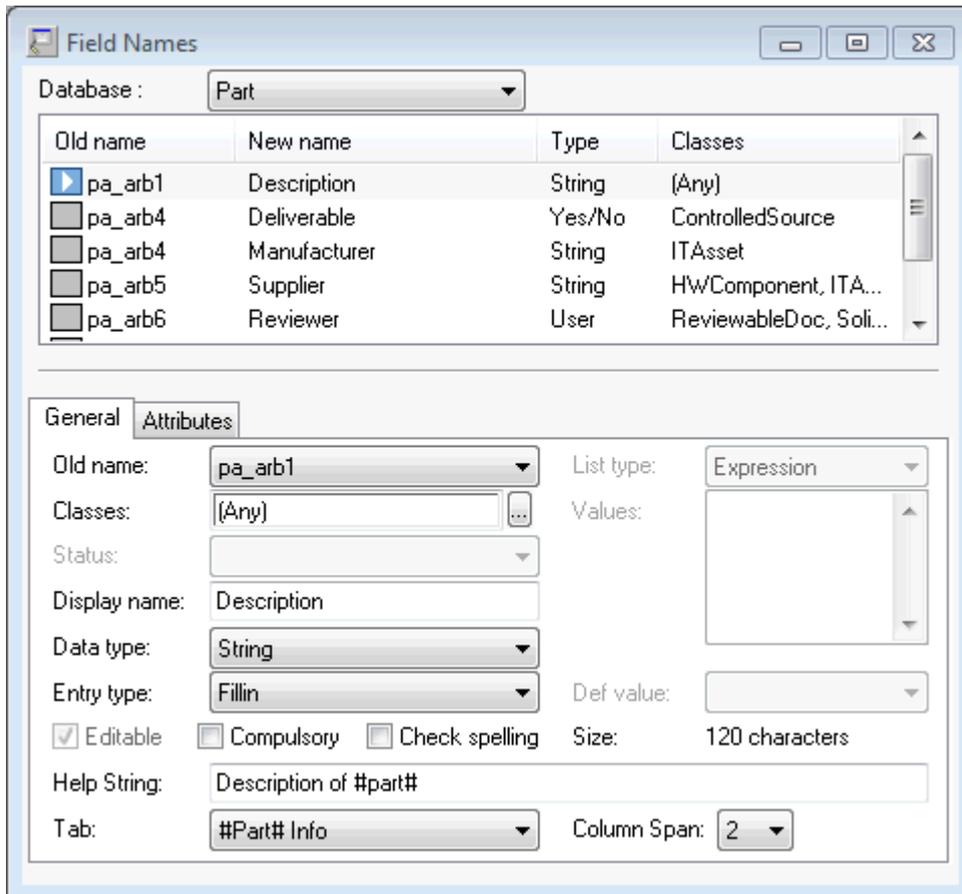
In addition various built in fields may be redefined according to site specific requirements

Also the **Other** option to the CR and Part **Status** commands may be used for different purposes for different statuses in the life-cycle.

In order to make the names more appropriate to their actual, usage facilities are provided to allow the names and value types to be specified for each of these fields.

The definitions of the use of these fields may be made to vary according to the class of the item to which they apply, i.e. different classes of item may have differently named fields.

Field name definitions may be specified using **GUI | Field Names** in ACCONFIG.



Select the **Database** whose fields you wish to define, and add/modify field name entries as required. Certain attributes may be disabled for built in fields, see [Redefining Built In Fields](#) for details.

*Note that any fields defined for a class with an Integration attribute will only will only be shown if the integration is enabled.*

Each field may specify the following information:

**Old name**

The real name of the field as used in ACCEL to identify the field, see [ACCEL Field References](#). This is made up of:

*db-prefix\_name*

**Class**

The name(s) of the class(es) to which this entry applies, or **(Any)** if it applies to all classes which do not have a specific definition. The same **Old name** may be defined more than once with the **Display name** varying depending on different **Classes**.

**Status**

Used only with entries for `pa_other` and `cr_other` — see below.

**Display name**

An arbitrary — hopefully more meaningful — *name* chosen by the site to be used in place of the **Old name**; obviously it should not clash with another name. This should specify the *name* part only (not the *db-prefix*). The same name may not be used for different classes.

**Data type**

This specifies the type of information that will be held in this field. Possible values are:

**Date:** this type causes data to be actually stored in internal date format which can be more easily used for date comparison purposes, but will be displayed in a user friendly date format. Date fields will also have appropriate operators available in the ACCEL condition editor

**Date/ Time:** similar to **Date** but includes the time as well, see above.

**Hyperlink:** this type causes text in an edit control to be treated as a hyperlink. The Entry Type for this type must be Fillin. The user can double click on the hyperlink to jump to the link. The user may edit 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.

**List:** this specifies that the field holds a list of items. This is used to ensure that the *list* operators are presented for this field in the ACCEL condition editor.

**Long String:** this specifies that the field may hold a *long* string. This is used to ensure that operators such as **Contains Text** and **Contains Words** which are appropriate to long strings are available in the ACCEL condition editor

**Numeric:** this specifies that the field holds a numeric value. Causes data to be actually stored in internal format which can be more easily used for comparison purposes, but will be displayed in a user friendly format. Values which are 'real' numbers are stored as <num>. <num> and are displayed with 2 decimal places and in the user's local format, (for instance '1.23' would be displayed as '1,23' in continental Europe). Numeric fields will also have appropriate operators available in the ACCEL condition editor.

**String:** this is the standard type

**User:** this specifies that the field holds a user name (i.e. a user logon id). **AllChange** will automatically use any **Full Name** specified in the [user registrations](#) for display purposes, but store the actual user id in the database field.

**User List:** this specifies that the field holds a list of user names (user logon IDs). **AllChange** will automatically use any **Full Name** specified in the [user registrations](#) for display purposes, but store the actual user IDs in the database field.

**Yes/ No:** this specifies that the field signifies a boolean value and may only hold the values **Yes, No**. These will be displayed in ACE according to the Entry Type selected as either a check box, (ticked for a **Yes** value, unticked for a **No** value) or a drop down list. If a field has no value (i.e. is blank) then this is interpreted as **No** or unticked.

### Entry type

This specifies the method/ type by which the data for the field will be entered. It affects how the field is displayed in ACE. Possible values are:

**ACCEL-prompt:** this will be displayed as an edit control with a . . . button leading to an **AllChange** list dialog, e.g. file list, part list, baseline list. The **AllChange** list dialog that is displayed is determined by the ACCEL code specified in the **Prompt**.

**Date picker:** this will be displayed as drop down calendar.

**Drop-down list:** this will be displayed as a combo-box and the values for the list may be specified - if the **Data type** is **Yes/No** then the values are Yes or No. Note that this allows a Yes/No field to be *optional*.

**Check Box:** this will be displayed as a check-box for **Data type** is **Yes/No**. Note that check boxes are by their nature *compulsory*.

**Fillin:** this will be displayed as a plain edit control

**Multi-select list:** this will be displayed as an edit control with a . . . button leading to a list from which multiple items may be selected. The values for the list may be specified. Including the value '(None)' in the list allows the field's value to be cleared.

**Read-only:** this will be displayed greyed/ disabled and is intended for use by fields which are to be automatically populated

**Single-select list:** this will be displayed as an edit control with a . . . button leading to a list from which a single item may be selected. This is very similar to a combo-box but the list values are not calculated until the . . . button is selected. This is useful where the list calculation is dynamic and may take some time, e.g. a list of baseline names. The values for the list may be specified. Including the value '(None)' in the list allows the field's value to be cleared.

#### **Editable**

Flag to indicate whether (or not) this is to be an editable list type of field (i.e. whether the value list is definitive or whether any arbitrary value may be entered).

#### **Compulsory**

Flag to indicate whether or not this field *must* be specified at the time an item is created to which this field applies. A compulsory **Read Only** field allows data to be input once at the creation of the object, thereafter it is greyed out and becomes read only. Compulsory fields are shown in **red** in Add dialogs. It is possible to display fields in viewers as compulsory based on an ACCEL condition, to indicate that a value should be entered to continue. See [Showing Fields as Compulsory for details](#).

#### **Check spelling**

This option specifies whether spell-checking should be enabled for the field. The option only applies to the Fillin entry-type. If checking is enabled, then any misspelled words are underlined with a red wavy line. A spell checking dialog may be opened from the field's right-click menu, or by pressing F7. See the Spell Checking topic in the User manual for details.

#### **List type**

This specifies whether the values for a list type field will be **Fixed** or a dynamically calculated **Expression**. If it is an expression then the **Values** should specify an ACCEL expression. If it is fixed then the **Values** should specify the valid values for the list. It is used with Combo, Multi-select and Single-select **Entry types**.

#### **Values**

May be used to offer the user possible values for list type fields. This may be a space/ newline separated list of values if the **List Type** is **Fixed**, or an ACCEL expression returning a list (or space separated string) of acceptable values if the **List Type** is **Expression**. If no values are specified then the list will be empty. Including the special value of (None) will allow users to blank out a field which has a value. The [get\\_window\\_field](#) and [get\\_window\\_item](#) functions may be used to allow a field's value list to depend on the value of other fields in the viewer/dialog. However, the field should use a single- or multi-select list entry type rather than a drop-down list, so that the expression is re-evaluated. Drop-down list expressions are only evaluated when initially populating the viewers/dialogs.

If a value starts with a + then this will be used as the default value when adding an item using this arbitrary field.

#### **Def Value**

May be used to select the default value to be used when a drop down list is specified. Only valid if using a **Fixed** value list.

#### **Prompt**

Should specify an ACCEL expression which cause a list dialog to be displayed and returns any selected value(s) on return. For example, `prompt_partlist`, `prompt_blinelist` etc. It may be preferable to invoke a user defined function which in turn calls one of these so that CANCEL can be handled.

#### **Help String:**

Specifies a help string to be shown as a tool tip when the mouse hovers over the field label or control and in the status bar when the cursor is in the control (e.g. when the user is typing into an edit control)

#### **Tab:**

Specifies the sub tab on the viewer on which this field should be displayed. Selecting **[Not shown]**

will prevent the field from being displayed at all. This might be useful, for example, for fields used for internal calculations only. Selecting **[New Tab...]** will cause a new tab to be created with a name of New Tab; this name may then be changed in the Field Tabs window. The position of the field within the tab may be specified in the [Field Tabs](#) Window

#### Column Span:

Specifies whether this field should span 1 or 2 columns on the field tab

#### Attributes:

Fields may have attributes associated with them which may be used to determine the behaviour of the field. There are some built-in attributes which may be dependent on certain features being enabled, and custom attributes can be added. Attributes have:

- A Name: this may be one of the built-in attributes if its dependencies are enabled or a custom attribute. To add a new attribute simply enter a new name
- A Value: this is optional

The following built-in attributes are defined:

- **SolidWorks Show in Taskpane:** This is dependent on the [SolidWorks Integration](#) being enabled. This has no value.
- **SolidWorks Synchronise:** This is dependent on the [SolidWorks Integration](#) being enabled. The value should be one of:
  - **To SolidWorks** - this will cause the field to be synchronised from **AllChange** to SolidWorks
  - **To AllChange** - this will cause the field to be synchronised from SolidWorks to **AllChange**
  - **Both Ways** - this allows the field to be synchronised either from SolidWorks to **AllChange** or from **AllChange** to SolidWorks depending on user selection at synchronisation time.
- **Revision Number:** Specifies that this field is a Revision Number field. The value for a Revision Number attribute should define the make up of the revision number using defined [Revision Counter](#)(s). A revision counter may be specified as %Revision Counter Name%, all other characters are used as they are. For example in the Inta-soft Standard configuration two Revision Number fields are defined:
  - **Issue No:** this has a revision number of %Major%.%MinorNumber%, the numbering sequence for this field will thus be 1.0, 1.1 etc.
  - **Revision:** this has a revision number of %Major%%MinorAlpha%, the numbering sequence for this field will thus be 1A, 1B etc

See also [Increment Revision action](#) for life-cycles.

As a special case, `pa_other` and `cr_other` may be used as **Old name**. These are not actual arbitrary field names; instead they are used to replace the **Other** field which normally appears when changing a part/CRs status by a meaningful **Display name** (and choice of values). In these entries the **Status** field is used to select what status the item must be going to for this replacement to occur. So, for example, an entry with **Old name** as `cr_other`, **Class** as `cr`, **Status** as `Rejected` and **Display name** as `Reason` would cause the **Other** label on the **Change CR Status** dialog to be changed to read `Reason` when the user selects to change a CRs status to `Rejected`.

When referencing an arbitrary field or renamed built field in ACCEL it may be referenced using its *oldname* or via its *displayname* with the appropriate database prefix. If the *displayname* contains space characters then these will be replaced with underscore (`_`) characters for naming purposes in ACCEL, e.g. `cr_Target_Release` for the **Target Release** field for CRs in the supplied arbitrary field definition for CR arbitrary field 8.

The definitions of the fieldnames is stored in the `fldnames.ac` configuration file in the project or system directory .

## Redefining Built In Fields

Certain built in fields may be redefined/renamed according to site specific requirements as shown below:

Field	Helpstring	Compulsory	Display Name	Entry Type	Data Type
cr_class	✓	✓	X	X	X
cr_summary	✓	✓	✓	✓	✓
cr_toppart	✓	✓	✓	✓	X
cr_ref	✓	✓	✓	✓	✓
cr_part-saffected	✓	✓	X	X	X
cr_bline-saffected	✓	✓	X	X	X
cr_crsaf-fected	✓	✓	X	X	X
cr_file-saffected	✓	✓	X	X	X
bl_class	✓	✓	X	X	X
bl_release-date	✓	✓	✓	X	X
bl_toppart	✓	✓	✓	✓	X
bl_comment	✓	✓	✓	✓	✓
pa_class	✓	✓	X	X	X
pa_part-saffected	✓	✓	X	X	X

In order for the condition editor to display the correct field names for built-in fields which have had their display names changed, the "Arbitrary Field" box should be ticked against the field in the [Condition Editor](#) window in ACCONFIG. (This is already set in the out-of-the-box configuration for supplied condition editor entries.)

Further, in order for the condition editor to show the correct entry type for built-in fields which have had their entry type changed, the value expression must be set to:

```
arbitrary_field_new_values(condedit_lhs, "UsedByAnyClass");
```

as per arbitrary fields. (This is already set in the out-of-the-box configuration for supplied condition editor entries.)

In order for the **Add/Remove Columns** dialog in ACE to offer the correct names for built-in fields which have had their display names changed, the fields should be marked with the **Arbitrary Field** flag in the [Browser Column Definitions](#) in ACCONFIG. If the built-in field's data-type may not be changed then the **Value** and the **Sort Value** may be specified, otherwise the **Value** should be set to the plain field name, and the Sort Value should be left blank, as for arbitrary fields. (This is already set in the out-of-the-box configuration for supplied browser column definitions.)

When writing reports or ACCEL code which displays a built-in field's name (for example, "Summary" or "Comment") you may display the field's new display name by calling a function, **FormatFldName**, defined in reportfunc.ac. The function is called with the field's old name, e.g. 'cr\_summary' as the first parameter, and the current item's class as the second, e.g. cr\_class, or empty if there is no current item. The field name must always be quoted as it is the field's name, and not its value, that needs to be passed to the

function. Note that the function should only be used for fields which can have their display names specified, as listed in the above table.

Calling the function on other fields will return an empty string.

For example:

```
call(FormatFldName, 'cr_summary', '')
```

will return the new display name for the `cr_summary` field defined for any class. If there is no entry defined for class 'any', then the default display name will be returned (in this case "Summary"). This is the way the function would be called to use as, for instance, a header row in a report, where the CRs listed could be of any class.

If the class is known at the time that the function is called, it may be called as follows:

```
call(FormatFldName, 'cr_summary', cr_class)
```

Here it can be used where the CR's summary is displayed just after its label. Where using the display name as a label, the function **FormatFldNameLabel** may be used which is the same as **FormatFldName** but the returned value has a colon (':') appended to it, providing it is not empty.

ACReport helps with this by providing Fields for Expression objects which use the functions above. The fields are defined for each of the built-in fields which may have their display name modified. There are two variants for each field: one for use in the header of a table, and one for use where the label is to the left of the field. They are named as per the following:

```
(<fieldname> Column Label) -for use in a table header
```

and

```
(<fieldname> Label) -for use elsewhere
```

where 'fieldname' is the out-of-the-box display name for the field.

For example, for the **Summary** field of a CR, the fields and their corresponding expressions would be as follows:

```
(Summary Column Label) => "call(FormatFldName, 'cr_summary', '')"
(Summary Label) => "call(FormatFldNameLabel, 'cr_summary', cr_class)"
```

See the supplied reports for more examples on the functions' usage.

### *Showing Fields as Compulsory*

In addition to using the Compulsory flag on a field to indicate that a value is required on creating an item, when showing a field in a viewer the field may be marked as compulsory at any stage. This allows, for instance, a field to be shown as requiring a value when the item on which it appears is in a particular status, or when assigned to a particular user. This is done by implementing a user-defined ACCEL function named **ShowFieldCompulsory**. The function returns 'true' if the field is to be marked as compulsory, else 'false'. This function is shipped in the out-of-the-box **acefunc.ac** function file.

The function takes two parameters: the field's old name and the viewed item's class. The database item shown in the viewer is the current record for that item type.

For example, to mark the **CR Summary** field as compulsory when a CR's status is **InWork**, the function would contain the following code:

```
local(oldname, class);
setvar(oldname, var(p1));
setvar(class, var(p2));
if var(oldname) == 'cr_summary' and cr_status == 'InWork' then
    return true;
endif;
```

Care should be taken when writing ACCEL code for this function that the code will not be too time-consuming to execute, as this will cause slowness in the display of viewers.

Note that this only affects the display of items in the viewers. The 'Add' dialogs continue to use the field's compulsory flag as defined in the field's definition in the Configuration Editor, while 'Alter' dialogs do not have a current record, and so are not able to determine whether a field should be displayed as compulsory.

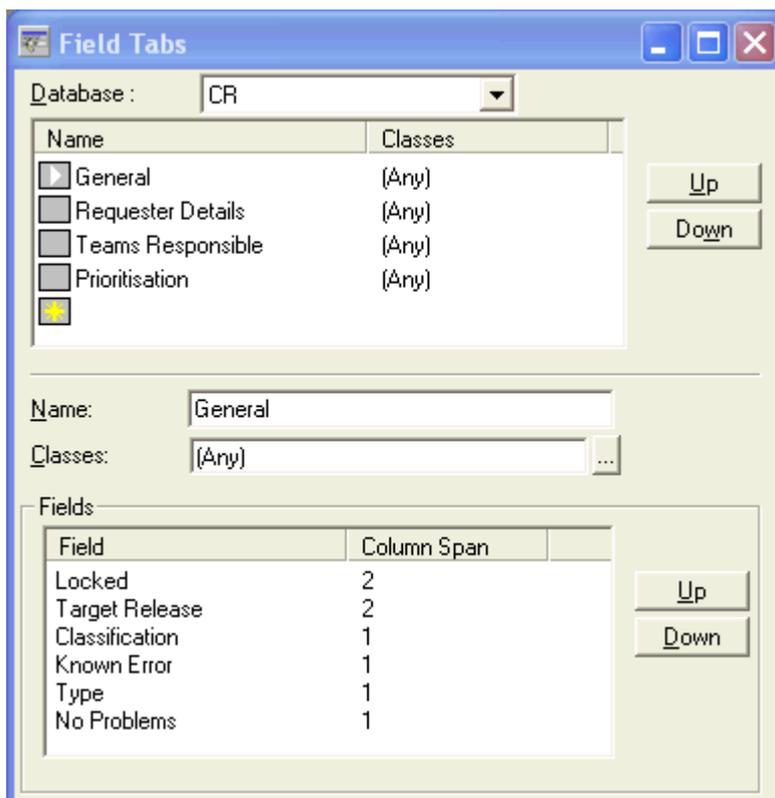
Also, marking a field as compulsory in this way does not enforce that a value is entered in that field. To enforce the entry of a value, ACCEL code should be added to the relevant command conditions to check for the value.

### Field Tabs

The sub tab on a viewer window in which arbitrary fields are displayed may be defined together with the order in which the fields are displayed, allowing site specific groupings of fields for different classes.

The tab on which a field appears is specified in the [Field Name Definition](#).

Field tabs may be specified using **GUI | Field Tabs**.



Select the **Database** whose fields tabs you wish to define.

The list at the top will show the Field Tabs defined for the database and the **Up** and **Down** buttons may be used to change the order in which these appear.

Each Field Tab has the following information:

**Name:**

The name of the tab. This will be the label for the tab on the appropriate Viewer. Nomenclature-mapping markers, may be used in the Name e.g. #Part#\_Info

**Classes:**

The classes for which this tab is defined. This must include all the classes for which fields on the tab are defined. The tab will not be shown if the item being shown is not one of these classes.

**Fields:**

A list of the fields which are to appear on this tab. Fields are added to a tab and their column span are specified in the [Field Name Definition](#). The order in which they appear on the tab may be determined here using the **Up** and **Down** buttons.

Fields are only shown on a tab if they are defined for the class of the items being viewed/added/alterd. Any tabs which either have no fields on them, or not defined for the current class are not shown.

Take care when placing fields on tabs to make sure that fields marked as compulsory will appear on a tab when creating an item for which the compulsory flag is relevant. The same applies to fields that have default values, as the values are pre-selected when the field is shown, so if the field does not appear, then the default value will not be set.

The definitions of the fieldnames is stored in the `fldtabs.ac` configuration file in the project or system directory

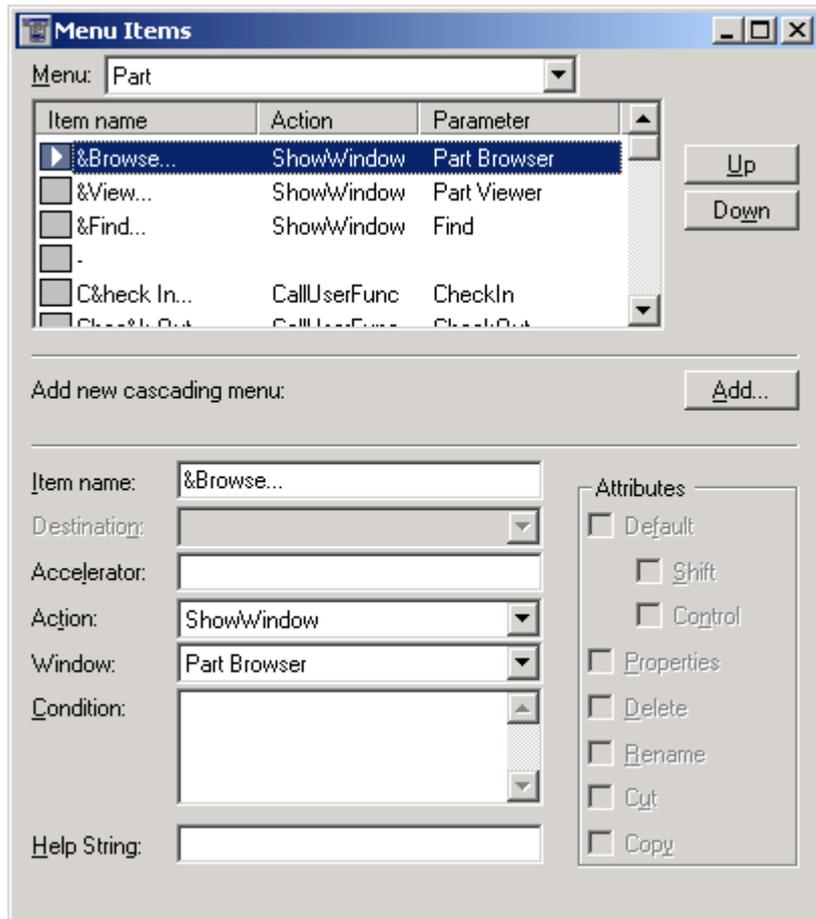
**Menu Items**

The options that appear on the menus in ACE and the actions that they perform are defined in the **Menu Item** definitions.

Different types of menu entries are defined:

- Pulldown menus from the menu bar
- Right click menus
- Right drag menus
- Cascade Menus

These may be tailored to site specific requirements using **GUI | Menu Item** in ACCONFIG.



- Plain Name:** for pulldown menus from the main menu bar. The name will match the text on the menu bar
- nameDrag:** for the menu produced when dragging an item from a window with the right mouse button down. The *name* indicates the window you are dragging *from*
- nameRight-Click:** for the context menu produced when you right click on an item.
- nameRight-ClickFree:** for the context menu produced when you right click on free space in a window

Where the *name* will be:

- *name* for a browser window such as **Part** for the part browser, **CR** for the CR browser.
- *name\_tab* to name the tab of a viewer window such as **Part\_Baseline** for the baseline tab of the parts viewer
- *name\_list-name* to name the list a viewer or browser window such as **CR\_Baselines\_Affected** for the Baselines Affected list on the Baselines tab of the CR viewer, or **Part\_Tree** for the subsystem tree list of the part browser window.

Select the menu you wish to modify and the list will show the items on that menu.

Each **Menu Item** has associated with it the following:

**Item Name:**

This identifies the item and is the text shown when the menu is displayed. The & (ampersand) character may be used to specify a mnemonic and . . . should be used to indicate that a dialog will be shown. An **Item Name** that simply comprises - (minus) characters is taken as a separator. *Note that each separator entry must be unique (i.e. it must have a different number of -'s).*

**Destination:**

This is used for **Drag** menu items to specify the destination **Drop** window or list.

**Accelerator:**

This should be used to specify an accelerator (or *shortcut*) key for a menu item. This may be specified as F2 . . . F12, CTRL+*key*, ALT+*key*. The CTRL and ALT may be used in combination and with SHIFT (e.g. CTRL+ALT+N would be CTRLALT+N).

**Action:**

This specifies the action that is to be taken when the menu item is selected, see [Menu Actions](#).

**Parameter:**

The title of this field will vary according to the **Action** selected and allows a parameter to be specified for the selected action, see [Menu Actions](#).

**Condition:**

This may specify an ACCELcondition which defines the condition under which the menu item is to be made available. It is recommended that most conditions are better specified associated with the *actions* they cause rather than the menu item; this means that the same condition does not need to be repeated for every place where an action may occur such as drop down menu, context menu, drag menu, toolbar button. Conditions for *actions* may be specified in **Menu Conditions**, see [Menu Item/ Toolbar Availability](#).

**Help String:**

This may specify a help string to be shown in the status bar when the cursor hovers over the menu item

**Attributes:**

These may be specified for **RightClick** and **Drag** menus:

- **Default** specifies whether the menu item is to provide the default action, i.e. the action on double click or left drag. This may be further refined to specify whether it should require a SHIFT or CTRL when dragging.
- **Properties** specifies whether the menu item performs the *properties* action to be executed on Alt+Enter.
- **Delete** specifies whether the menu item provides the *delete* action to be executed on Del
- **Rename** specifies whether the menu item provides the *rename* action to allow rename to take place in place in browse lists.
- **Cut** specifies whether the action is a valid **Paste** operation from a previously performed **Cut to Clipboard**. This is only valid for drag'n drop operations allowing items to be cut to the clipboard in one window and pasted in another according to the specified **Destination** and the **Source** (denoted by the name of the menu item).
- **Copy** specifies whether the action is a valid **Paste** operation from a previously performed **Copy to Clipboard**. This is only valid for drag'n drop operations allowing items to be copied to the clipboard in one window and pasted in another according to the specified **Destination** and the **Source** (denoted by the name of the menu item).

As well as defining the contents of menus and the actions that menu items perform it is also possible to create new cascade menus. Select the **Add** button from the **Add new cascading menu** section of the **Menu Item** window. Specify the name for the new menu when prompted and the menu will be added to the list of **Menus**. **Menu Items** may then be added to the cascade menu in the normal way. The cascade menu must also be added as an **Item** to an existing menu in order for it to be displayed.

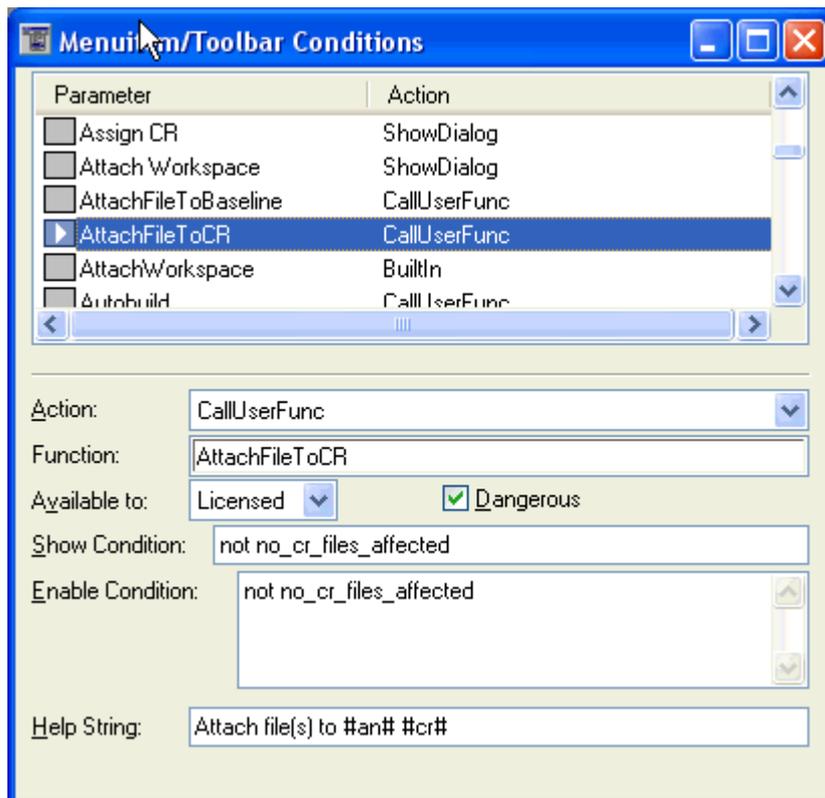
The menu item definitions are stored in the `menuitem.ac` configuration file in the project or system directory .

**Menu Item/ Toolbar Availability**

The items which appear on menus and toolbar buttons may be disabled (or greyed) when they are not valid to perform the action that will occur on selecting the menu item or toolbar button. Furthermore, menu items may be removed/hidden entirely if not required.

The conditions under which these are disabled/removed is specified according to the action that will be performed. Disabling an action will cause *all* menu items (pulldown, right click and drag) toolbar buttons and buttons on viewers which perform the action to be disabled. Hiding/removing a menu item will cause *all* menu items (pulldown, right click and drag) which perform the action to be removed/hidden.

The conditions may be tailored to site specific requirements using **GUI | Menu Conditions** in ACCONFIG .



This shows a list of all the *unique* actions which may be performed. The action is uniquely identified by the **Action** type and the **Parameter** which defines the precise action to be taken. The list is therefore sorted by **Parameter** and the **Action Type** is shown in the second column.

Each *unique* action has associated with it the following:

**Action:**

This identifies the type of action and may be one of those described in [Menu Actions](#)

**Parameter:**

This specifies the parameter to the action and is what uniquely identifies the action. The parameter type (and control label in the Menu Condition window) will vary according to the action, see [Menu Actions](#)

**Available to:**

This specifies what type of user should have access to this action. If a user is not of the specified type then the menu item or toolbar button which invokes the action will be removed. The type may be one of:

- **All:** all users are permitted this action
- **Licensed:** only licensed (Full or CR-only) users are permitted this action
- **Full:** users which are *not* CR-only users are permitted this action
- **Admin:** Users who are **AllChange** administrators are permitted this action
- **None:** no users are permitted this action. This is useful for totally disabling a functionality

**Dangerous:**

This specifies whether this menu option will cause database changes. If it does it should be flagged

as **Dangerous** (this is the default case). Only items which perform read-only operations should be flagged as not dangerous (e.g. reports). This flag is used to determine whether a check for unsaved changes in viewer windows needs to be made before the operation proceeds.

#### Show Condition:

This specifies any ACCEL condition that must evaluate to *true* in order for the menu item for the action to be shown. This may be used, for example, to remove menu items for a feature which has been disabled. The **Show Condition** is evaluated just once on start-up of **AllChange** and menu-items hidden will remain hidden for the duration of the session. It is advisable that **Show Conditions** specified should be simple ACCEL as they will affect the start-up time of **AllChange**. It is also advisable not to use ACCEL which will make database accesses, as these will also affect start-up time. A suitable condition to remove menu items associated with the instances feature if this is not used would be:

```
not no_instances
```

#### Enable Condition:

This specifies any ACCEL condition that must evaluate to *true* in order for the action to be permitted i.e. the menu item or toolbar button to be enabled. This may be used, for example, to disable menu items which are not appropriate for browsers in folding view (such as **Print**). In this case a suitable condition would be:

```
not match_wild(get_selected_list(), "*_Folding")
```

#### Help String:

This may specify a help string to be displayed in the status bar when the cursor hovers over a menu item or toolbar button which invokes the specified action

The menuitem availability information is stored in the menucond.ac configuration file in the project or system directory .

### Menu Actions

Menu items and toolbar buttons may perform a number of predefined actions. The menu items which invoke those actions and the conditions under which the actions are available are determined by the **Menu Item** definitions, [Menu Items](#) and the **Menu Condition** definitions, [Menu Item/ Toolbar Availability](#).

The available actions are:

<b>BuiltIn</b>	Built in commands and actions
<b>CallUserFunc</b>	Call the specified user defined function
<b>ReadItem</b>	Read same/ next/ previous item for the current upfront browse or view window
<b>SetToggle</b>	Tick/ Untick menu item. Used for <b>Autoupdate</b> and <b>ShowIssueState</b> etc.
<b>SetUserToggle</b>	Tick/ Untick menu item and set/ clear user-defined variable; the variable may then be tested to implement the toggle action
<b>ShowCascade</b>	Show specified cascading menu
<b>ShowDialog</b>	Show specified dialog
<b>ShowWindow</b>	Open/ show specified window
<b>ShowNewWindow</b>	Open a new instance of the specified window

Each action may have a parameter associated with it to further define the action required.

The parameters for the **BuiltIn** action specifies the exact action required. This may be one of:

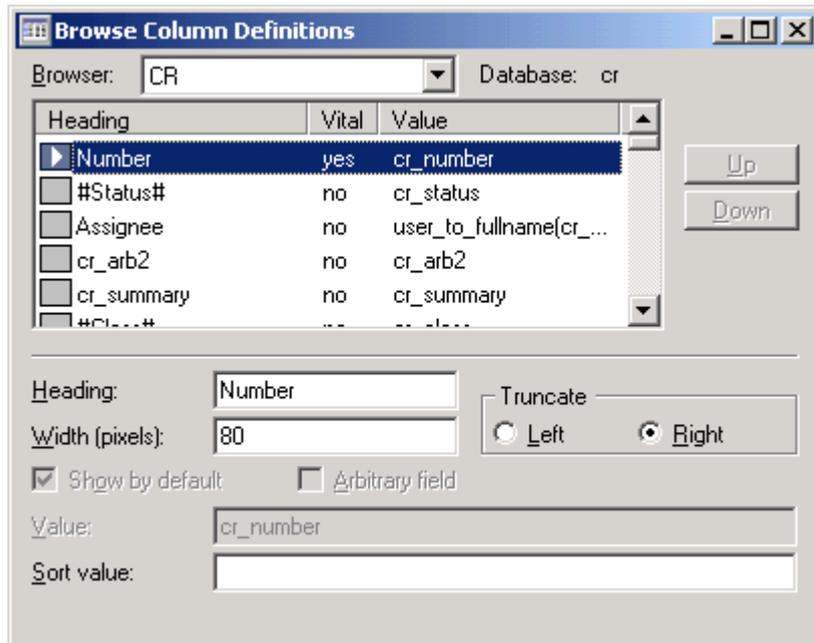
<b>AddBlinesAffected</b>	Prompt for and add baselines to baselines affected list of a cr
<b>AddBlinesSolved</b>	Prompt for and add baselines to baselines solved list of a cr
<b>AddBlineDetail</b>	Prompt for and add parts to baseline
<b>AddCRsAffected</b>	Prompt for and add CRs to CRs affected list of a cr
<b>AddBaselineFilesAffected</b>	Prompt for and add a file affected (attachment) to a baseline
<b>AddFilesAffected</b>	Prompt for and add a file affected (attachment) to a cr
<b>AddMetaBlineDetail</b>	Prompt for and add baseline to a meta-baseline
<b>AddPartPartsAffected</b>	Prompt for and add parts to parts affecting list of a part

<b>AddPartsAffected</b>	Prompt for and add parts to parts affecting list of a cr
<b>AddVersionsSolved</b>	Prompt for and add versions to versions solved list of a cr
<b>AttachWorkspace</b>	Attach to selected workspace
<b>ChangeWorkingPart</b>	Change working part to selected part
<b>ClearAceSettings</b>	Clears the ACE registry settings when ACE next quits.
<b>CopyToClipboard</b>	Copy selected text to clipboard
<b>CutToClipboard</b>	Cut selected text to clipboard
<b>DeleteBaseline</b>	Delete selected baseline
<b>DeleteBlinesAffected</b>	Delete selected baseline affected
<b>DeleteBlinesSolved</b>	Delete selected baseline solved
<b>DeleteBlineDetail</b>	Delete selected detail from baseline
<b>DeleteCR</b>	Delete selected CR
<b>DeleteCRsAffected</b>	Delete selected CR affected
<b>DeleteBaselineFilesAffected</b>	Delete selected file attachment
<b>DeleteFilesAffected</b>	Delete selected file attachment
<b>DeleteMetaBlineDetail</b>	Delete selected meta-baseline detail
<b>DeleteMonitor</b>	Delete selected monitor
<b>DeletePart</b>	Delete selected part
<b>DeletePartPartsAffected</b>	Delete selected parts from parts affecting a part
<b>DeletePartsAffected</b>	Delete selected parts from parts affecting a cr
<b>DeleteVersionsSolved</b>	Delete selected versions from versions solved by a cr
<b>DoHelp</b>	Show help
<b>Exit</b>	Exit <b>AllChange</b>
<b>FileProperties</b>	Show properties of selected file
<b>PasteFromClipboard</b>	Paste text from clipboard
<b>RefreshWindows</b>	Refresh all windows
<b>SelectAllListItems</b>	Select all items in list
<b>SetRole</b>	Set current role to selected role
<b>SetShowFoldingView</b>	Allows the Folding View to be switched on and off
<b>SetSaveSettings</b>	Set save settings on exit flag on/off
<b>ShowTBDialog</b>	Show toolbar configuration dialog
<b>UndoEdit</b>	Undo last edit
<b>UpdateItem</b>	Save changes made in current viewer
<b>UpdatePartsSolved</b>	Assign default version of selected parts affecting a CR as the version solved by the CR (Solved button on CR parts affecting list)

## Browsers

The contents of the various browser lists in ACE may be tailored by each user using the **Add/ Remove Columns** facilities in ACE. The **AllChange** administrator may define what information is shown by default and the choices available to the user in the **Add/ Remove Columns** dialog by modifying the definitions.

In ACCONFIGselect **GUI | Browser**.



The browser that is to be modified is selected as the **Browser**.

The Browsers which may be modified are:

#### Baseline

Defines the contents of the **Baseline Browser** window. The first column must be the baseline name.

#### Baseline CRs Affecting

Defines the information displayed in the **CRs Affecting** tab when viewing a baseline. The first column must be the CR number.

#### Baseline CRs Solved

Defines the information displayed in the **CRs Solved** tab when viewing a baseline. The first column must be the CR number.

#### Baseline Detail

Defines the contents of the **Parts** tab when viewing a normal (i.e. part-) baseline. The first column must be the partname.

#### Baseline Files Affected

Defines the contents of the **Attachments** tab when viewing a baseline. The first column must be the filename.

#### Baseline List

Defines the contents of the baseline browser list dialog. The first column must be the baseline name.

#### Baseline Meta Detail

Defines the contents of the **Baselines** tab when viewing a meta-baseline. The first column must be the baseline name.

#### Baseline Meta List

Defines the contents of the **Meta** tab when viewing a baseline. The first column must be the baseline name.

**Baseline Status Log** Defines the contents of the status log tab on the baseline viewer window.

#### CR

Defines the information shown in the **CR Browser** window. The first column must be the CR number.

**CR Baselines Affected**

Defines the information shown in the **Baselines Affected** list in the **Baselines** tab in the CR viewer. The first column must be the baseline name.

**CR Baselines Solved**

Defines the information shown in the **Baselines Solved** list in the **Baselines** tab in the CR viewer. The first column must be the baseline name.

**CR CRs Affected**

Defines the information shown in the **CR Affects** list in the **CRs** tab in the CR viewer. The first column must be the CR name.

**CR CRs Referring**

Defines the information shown in the **CRs Referring** list in the **CRs** tab in the CR viewer. The first column must be the CR name.

**CR Files Affected**

Defines the contents of the **Attachments** tab when viewing a CR. The first column must be the file-name.

**CR List**

Defines the information shown in the **CR Browser List** dialog. The first column must be the CR number.

**CR Parts Affected**

Defines the information shown in the **Parts Affected** list in the **Parts** tab in the CR viewer. The first column must be the part name.

**CR Status Log**

Define the contents of the status log tab on the CR viewer window.

**CR Versions Solved**

Defines the information shown in the **Versions Solved** list in the **Parts** tab in the CR viewer. The first column must be the part name.

**Instance**

Defines the information shown in the **Instance Browser** window. The first column must be the instance name.

**Issue**

Defines the information shown in the **Issue Browser** window. The first column must be the part-name and the second must be the workspace .

**Monitor**

This defines the contents of the **Monitor Browser** window. All columns are compulsory.

**Part**

Defines the information displayed in the **Part browser** window. The first column must be the part name.

**Part Baseline**

Defines the information displayed in the **Blines** tab when viewing a part. The first column must be the baseline name.

**Part CRs Affecting**

Defines the information displayed in the **CRs Affecting** tab when viewing a part. The first column must be the CR number.

**Part CRs Solved**

Defines the information displayed in the **CRs Solved** tab when viewing a part. The first column must be the CR number.

**Part Issue**

Defines the information displayed in the **Issues** tab when viewing a part. The first column must be the partname and the second must be the workspace.

**Part List**

Defines the information displayed in the **Part List** dialog. The first column must be the part name.

**Part List Tree**

Defines the information displayed in the **Part List Tree** dialog. The first (and only) column must be the part name.

**Part Parts Affected**

Defines the information displayed in the **Part Affected** list on the Parts tab of the Part Viewer. The first column must be the part name.

**Part Parts Referring**

Defines the information displayed in the **Part Referring** list on the Parts tab of the Part Viewer. The first column must be the part name.

**Part Status Log** These define the contents of the status log tab on the part viewer window.

**Part Tree**

Defines the information displayed in the **Part Tree** of the part browser. The first (and only) column must be the part name.

**Role**

This defines the contents of the **Role Browser** window. The first column must be the role name.

**Version List**

This defines the contents of the **Select Version** dialog (used from the part viewer version tab). The first column must be the version name.

**Workspace**

This defines the contents of the **Workspace Browser** window. The first column must be the workspace name.

In addition, for browsers that support a *folding* view there is an entry for specifically for the folding view. This will have the same name as the browser but with **(Folding)**. The columns specified here are not used as information from the standard view browser columns will be used instead. The folding entry simply enables the folding view for that browser.

Note that with the use of [nomenclature mapping](#) browsers may appear with different naming to the underlying **AllChange** browser name in ACE; for example the out-of-the-box configuration maps issue to check-out, thus the **Issue** browser will show as **Browse Check-outs**. The underlying **AllChange** browser name, however, remains Issue and this is what must be used in configuring browser columns.

For the selected browser, the columns available to the user from ACE are shown in the list and the details of the currently selected column will be shown in the details section of the window.

Note that whilst, by default, all arbitrary fields are in the list, only those which are defined/used will be presented to the user.

Each column entry has the following information associated with it:

**Heading**

Specifies the title for this column.

**Width**

Specifies the initial width of the column in pixels.

**Truncate**

Allows left or right truncation of the column value to be specified if it does not fit in the current width allowed.

**Show by default**

Determines whether this column is shown as a default column.

**Arbitrary field**

Determines whether the column represents an arbitrary field value. Arbitrary fields should have the **Heading** as the field name for the arbitrary field which will be presented as the defined name if the arbitrary field is used. For built in fields which may have their name changed this should be selected.

**Value**

Specifies the *display* value for the column; i.e. the contents of the column as an ACCELexpression. The value of each expression up to the width specified will be shown truncating the column contents as appropriate. If it is an arbitrary field then the value specified should be the field name and the control will be disabled.

**Sort Value**

Should specify the value to be used for sort purposes when the column is clicked on (as opposed to the *display Value*); this is not available for the CR browser. This should be used for cases where the display value is not appropriate for sort purposes, e.g. for date fields. If it is an arbitrary field then no sort value may be specified and the control will be disabled.

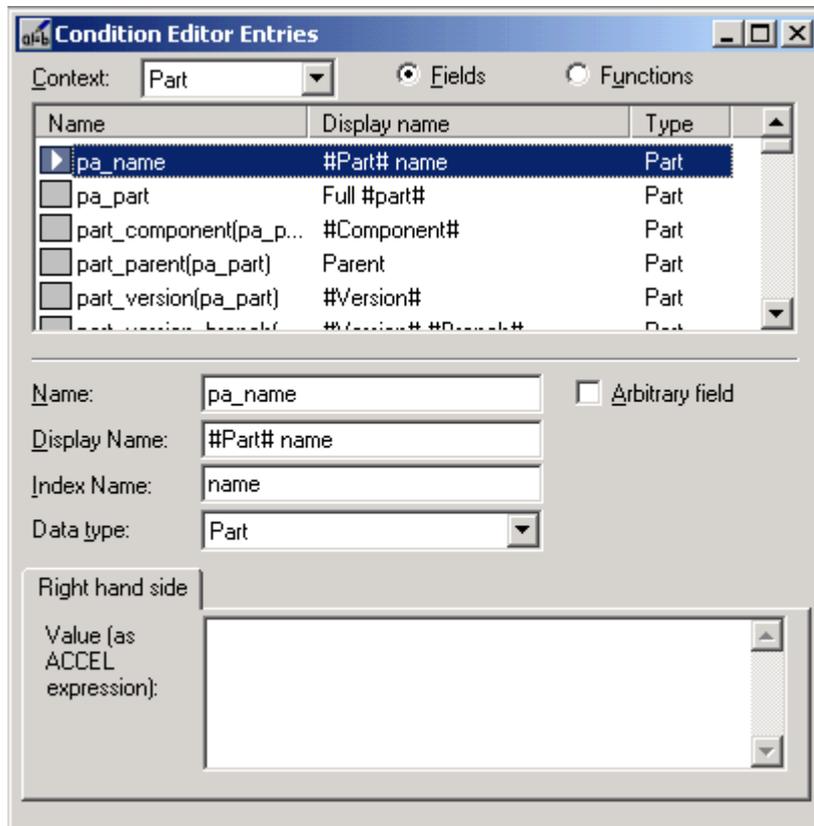
Some of the above information may not be modified for certain fields, these are the key fields for the database being browsed.

The browser definitions are stored in the `browser.ac` configuration file in the project or system directory .

**ACCEL Condition Editor**

The contents of each field/function/value list in the ACCELcondition editor may be tailored to site specific requirements. You may, for example, wish to add user defined functions to the list of available functions in the condition editor.

In ACCONFIGselect GUI | **Condition Editor**



The contents of the field/ function lists available in the ACCEL condition editor vary according to the **Context**, e.g. if the condition is to be used for reporting on parts then only part fields and functions appropriate to parts will be presented in combo box lists.

Furthermore, for each context there are the **Fields** available and the **Functions** available.

To modify, therefore, the fields available in the part condition editor (e.g. used in the part browser window in ACE), select the **Part** context and the **Fields** radio button.

The list in the **Condition Editor Entries** window will show all the fields/ functions available for the selected context; these form the contents of the field/ function combo box in the condition editor in ACE.

For each field the following information is held:

#### **Name**

This should specify the name of the field as known to ACCEL.

#### **Arbitrary field**

If this is selected then the field is taken to be an arbitrary field. This should be selected for built in fields which have been/may be renamed as well as arbitrary fields, see [Field Name Definitions](#).

#### **Display name**

This should specify how this field should be presented in the field list in the condition editor in ACE. This is also used to present *index* filter names in browsers and report dialogs. For arbitrary fields the **Display name** is not used, instead the display name used will be that defined in the **Field Name** definitions.

#### **Index name**

This should specify the index name to be used for this field. This is used to ensure that the correct indexing is used when an *index* filter is selected in a browser.

#### **Data type**

This should specify the type of data that the field holds. This is used for determining the operators that should be available for fields of this data type. Note that this information is not used for arbitrary fields. For any *defined* arbitrary field this information will be taken from the **Field Name** definition instead. Undefined arbitrary fields will not be presented in ACE in the condition edit field lists. The data type may be one of:

**Date:** should be used for fields which hold dates.

**List:** should be used for fields which hold a list of items.

**Long String:** should be used for fields which hold a *long* string.

**Part:** should be used for fields which hold part names.

**String:** should be used for fields which hold a standard string

**Truth:** should be used for fields which hold a boolean value.

**User:** should be used for fields which hold user names or groups

#### **Right hand side**

This allows an ACCEL expression to be defined which should return a list of the valid values for the **value** list in the condition editor in ACE. For builtin fields which may have their data type modified this should be treated the same as arbitrary fields with `arbitrary_field_new_values(condedit_lhs, "UsedByAnyClass");` as the expression.

For each function the following information is held:

#### **Name**

This should specify the name of the function.

#### **No. parameters**

This should specify the number of parameters that the function takes, this may be a value from 0-3.

## Parameters

Up to three parameters may be passed to the function according to the value specified for **No. parameters**. For each parameter the following information should be supplied:

**Label:** this should specify the label to be used for the parameter. This will appear above the combo box where the parameter value may be supplied by the user in the condition editor in ACE

**Value:** this allows an ACCEL expression to be defined which should return a list of the valid values for the **parameter** combo-box in the condition editor in ACE.

The condition editor information is stored in the `condedit.ac` configuration file in the project or system directory .

## Nomenclature Mapping

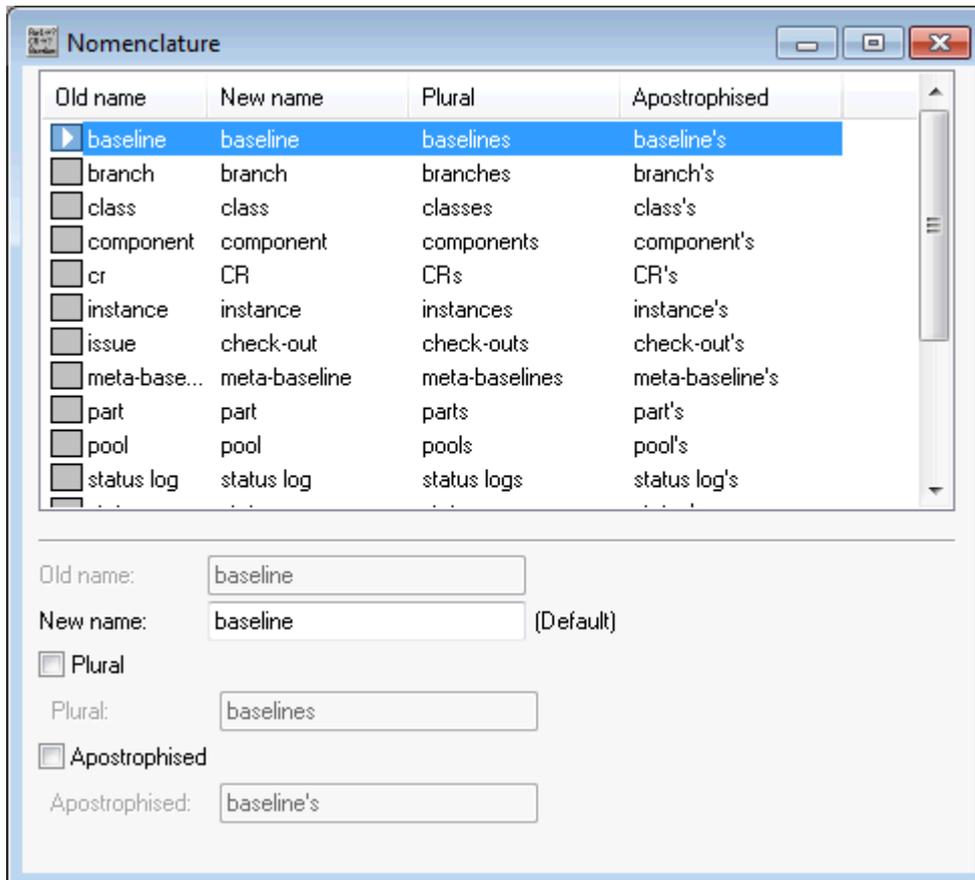
The nomenclature mapping facilities allow the terminology used in **AllChange** to be changed/mapped to site specific terminology.

The terms that may be mapped are:

- baseline
- branch
- class
- component
- cr
- instance
- issue
- meta-baseline
- part
- pool
- status log
- status
- subsystem
- version
- vote
- voter
- workspace

*Note that the out-of-the-box configuration includes a mapping of issue to check-out.*

In `ACCONFIGselect` **GUI | Nomenclature**



The terms that can be mapped are shown in the list in the Nomenclature window. For each term the following may be specified:

**New Name:**

This should specify the new name (or term) to be used instead of the Intasoft default term. The new name may contain spaces if desired (e.g. change request). The new name should normally be entered in lower case unless any letters should only ever appear in upper case. For example if the replacement for "CR" is to be "Change Request", then it would normally be entered as "change request", unless it should always appear as "Change Request".

The maximum length of the new name is 40. Be aware that although every effort is made to auto-size labels in **AllChange** to fit the chosen mapping, very long replacements may not appear in full in all cases.

**Plural:**

If the **Plural** check box is selected then a pluralised form of the new name may be specified. This overrides the default pluralisation which will be shown when the new name is specified

**Apostrophised:**

If the **Apostrophised** check box is selected then an apostrophised form of the new name may be specified. This overrides the default apostrophisation which will be shown when the new name is specified.

All Out-of-the-box Menu items, Browser columns, Condition Editor fields, built-in messages, ACCEL code, reports etc support any nomenclature mappings specified.

The nomenclature mapping information is stored in the `nomenmap.ac` configuration file in the project or system directory .

## Mapping Markers

Any customised/site specific ACCEL code which displays text to users should be formatted such that the mappable terms are mapped to the current specified nomenclature. The accel functions [map\\_nomenclature](#), [error\\_map](#), [echo\\_map](#) and [trace\\_echo\\_map](#) together with ACCEL functions `MessageMap`, `MessageEchoMap` and `show_info_list_map` defined in `utility.ac` allow the conversion of text containing mapping markers to be translated to the appropriate nomenclature.

Mapping markers are used to identify which word/term in a text string to map, and also the case and pluralisation or apostrophisation. These markers are identified by being enclosed in # characters. Furthermore, to assist in constructing sentences, there is a special marker of `#an#`, which is substituted for **a** or **an** depending on whether the following word starts with a vowel.

The case, pluralisation and apostrophisation are specified as follows:

- If the first character of the marker is in lower case, e.g. "`#cr#`", then the case of all replacement words is left as it is e.g. "change request".
- If the first character is in upper case, but the next character is lower case, e.g. "`#Cr#`", then each of the replacement words will have its first character converted to upper-case, while the remainder of each word is left in its current case, e.g. "Change Request".
- If the first two characters of the marker are in upper-case, e.g. "`#CR#`", then the entire replacement is converted to upper-case, e.g. "CHANGE REQUEST".
- If the marker contains an underscore (`_`) character then the marker is mapped to the plural form of the replacement word(s)
- If the marker contains an apostrophe (`'`) then the marker is mapped to the apostrophised form of the replacement word(s).

The following table summarises the marker mappings using the `cr` term as an example:

cr Marker Mapping	Description	Example Mapped Result
<code>#cr#</code>	'normal' case	CR, part, baseline, status log
<code>#Cr#</code>	initial capitals	CR, Part, Baseline, Status Log
<code>#CR#</code>	upper-case	CR, PART, BASELINE, STATUS LOG
<code>#cr_s#</code>	'normal' case plural	CRs, parts, CRs, baselines
<code>#Cr_s#</code>	initial capitals plural	CRs, Parts, Baselines
<code>#CR_s#</code>	upper-case plural	CRS, PARTS, BASELINES
<code>#cr's#</code>	normal apostrophised	CR's, part's, baseline's
<code>#Cr's#</code>	initial capitals apostrophise	CR's, Part's, Baseline's
<code>#CR's#</code>	upper-case apostrophised	CR'S, PART'S, BASELINE'S
<code>#an#</code>	'a' or 'an'	
<code>#An#</code>	'A' or 'An'	
<code>#AN#</code>	'A' or 'AN'	

Some example sentences showing the usage of the markers are below:

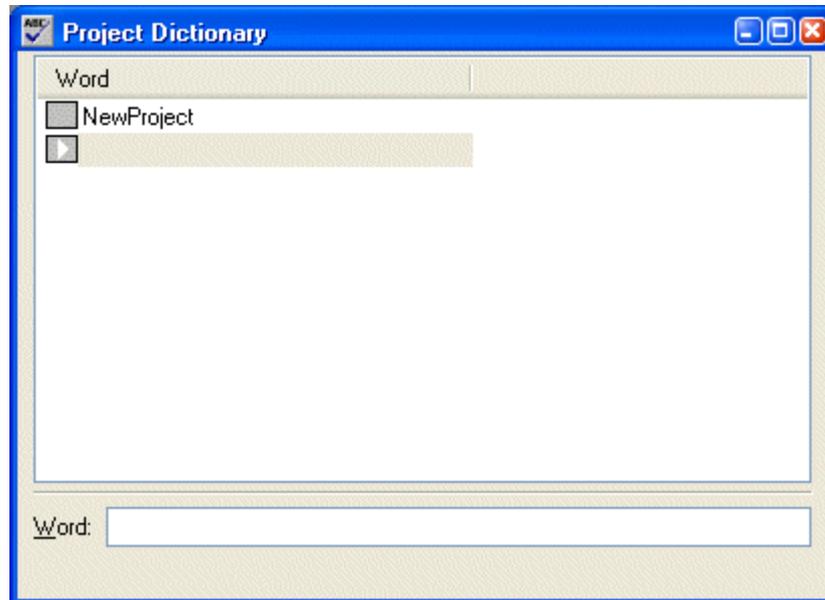
- "We add `#part_s#` to `#an# #baseline#`, and check them out to `#an# #workspace#`."
- "We do not have `#an# #status log#` entry for this `#part's# #status#`."
- "`#Baseline_s#` can appear in one or more `#meta-baseline#`."
- "This `#part#` requires `#an# #cr#` in order to create a new `#version#`."

Note that where the sentence would have contained the word 'CR', it is usually replaced with `#cr#` or `#Cr#`, as, if the mapping is specified as, for example, 'incident', it will usually want to appear in lower-case ('incident'), or initial capitals ('Incident'), rather than all upper-case.

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

### Project Dictionary

The project dictionary may be edited in the AllChange Configuration Editor, so that words specific to your project may be added. This can be used to prevent any non-English words from being flagged as misspelled words when performing spell-checking operations in AllChange.



### Intercepting Viewer Buttons

Some of the buttons appearing on Viewers may be intercepted, to display site-defined dialogs instead of the built-in dialogs.

### Intercepting Standard Dialogs

To intercept the dialogs, an ACCEL function called **ShowDialogCallback** should be defined. This function is defined in the out-of-the-box configuration in acefunc.ac. If defined, the function is called before showing the dialog opened by the relevant button. This allows the behaviour to be modified, or replaced. The function takes one parameter: the name of the dialog. The function should return *true* if the dialog call is completely handled, or *false* to let AllChange perform the default action.

The buttons which may be intercepted, along with the parameter passed to **ShowDialogCallback** are as follows:

CR Viewer	Assign button on General tab	Assign_CR
-----------	------------------------------	-----------

For example, to replace the built-in Assign CR dialog, the function is defined, and something like the following can be used:

```
ShowDialogCallback
# p1: the dialog name, in the format as specified for ShowDialog menu
  items
action
  local(dialog);
  setvar(dialog, var(p1));
  if var(dialog) == "Assign_CR" then
```

```

local(cr);
setvar(cr, get_selected('CR'));
call(MyAssignCRDialog);
return true; # we've handled it
endif;
return false; # do the default action
end

```

### Intercepting Assignee list

In addition to being able to intercept the dialogs, the list of users shown in the **Assign CR** dialog can be modified in a similar way. An ACCEL function named **GetAssignCRUserList** is called to get the list of users for the drop-down list in the dialog. The function is defined by default in `acefunc.ac`, and may be modified to tailor the user-list, for instance, to exclude certain users.

## Register Definition Files

Register files are called `register.ac`. The directory associated with each workspace may contain this file. It specifies which Pools are to be used in build processes, and it also allows the specification of what the default versions of parts should be for the workspace.

Register files need to be set up and understood by normal users; accordingly they are described in the **All-Change** User Manual.

## Creating Monitor Events

The `baseline`, `cr` and `part` monitor events may be used to place a monitor on an object of the corresponding type for any event required by the project. The event must be configured to be monitored by using the ACCEL function `check_monitor()` and the monitor's arbitrary field 1 to specify the specific event required.

The supplied `commands.ac` file implements a few monitors for the **baseline**, **cr** and **part** events: these are documented in the user manual.

In order to add additional events the **AllChange** administrator should perform the following operations:

1. Add a `check_monitor()` call to the end of the actions section of the appropriate entry in the `commands.ac` file. For example, to allow a monitor to be placed for CRs being deleted, `check_monitor(cr_number, 'cr')` should be added to the end of the `deletecr` action in the `commands.ac` file. Whenever a CR is deleted this will search the monitors database for any monitors on the same CR number (or any) for event `cr`.
2. Add the command name to be specified as the monitor `arb1` field as a valid value in the `monitor` entry condition in the `commands.ac` file, e.g. add `deletecr` to the list of valid "arb1" values in the `monitor` entry condition in the `commands.ac` file.
3. If the monitor event does not correspond directly to an **AllChange** command the `monitor_action` entry in the `commands.ac` file will need to be modified to allow for a special arbitrary field value to denote the event. For example, if the user wishes to monitor for changes to the CR priority (which is the `cr_arb2` field) an `alterpriority` event needs to be coded for in the `monitor_action`.
4. The new event should be added to the `fldnames.ac` file as a valid value for the `mo_arb1` field.
5. Tell the user that to place a monitor for the required event, the generic monitor event appropriate should be selected, and the appropriate `Command` or `arb1` value should be selected when placing the monitor. For example, for monitoring `deletecr` the monitor event should be set to `cr` and `arb1` to `deletecr`.

Now whenever a matching monitor is found and triggered in step 1 the code in `monitor_action` is executed. The ACCEL variable `command_name` will be set to the name of the current command; any

monitor whose "arb1" field is the same as the current command name or a specially coded action is taking place will send mail to the monitor placer.

Finally, as an aside, there may be some confusion over when to use monitors versus when to just put a `call(SendMail, ...)` in action sections of the `commands.ac` file. Monitors allow individuals to choose to be notified when events happen on certain (or any) items; they require individuals to actively place a monitor on those events/ items they are interested in. Putting a `call(SendMail, ...)` in a command's actions always mails a user whenever that event occurs on any item; they require no activity from the user. Monitors are *much* less efficient: an entry must be held in the monitors database for every item/ event/ user combination monitored and this database must be searched whenever any user performs an action which is automatically monitored or has a `check_monitor()` in its action. So the mailing of a user when a CR is assigned to him should be accomplished via a `call(SendMail, ...)` in the actions for `assigncr` since we want it to always happen and we know whom to mail. But if, say, a user wishes to know when a particular CR is reassigned among his colleagues this is best achieved by him placing his own monitor on `assigncr` for that CR.

## Administrator Facilities

### ACE Facilities for Administrators

Various **AllChange** functions and facilities are available in ACE which are for use by **AllChange** administrators. Some functions may be restricted to administrator access only. Precisely which functions are classified as administrator only is site modifiable according to the **Menu Condition** definitions, see [Menu Item/ Toolbar Availability](#).

The *out-of-the-box* configuration provides the following as administrator only functions:

#### Save Shortcuts

Administrators will have this option on the shortcut bar and folder list right click context menus. If selected the current shortcut bar and folder list settings will be saved to a configuration file in the project directory. These settings will then be default for users.

#### Who

It may be useful to determine what users are currently using **AllChange**. An ACCEL function **Who** is available from the **Function** menu in ACE. See the Who chapter of the **AllChange** Reference Manual.

#### Send Mail to AllChange Users

It is often useful as an **AllChange** administrator to mail all **AllChange** users, for example when a change to the configuration has been made.

An ACCEL function **Send Mail To AC Users** is available from the **Functions** menu.

This will prompt for a message and a subject and send this mail to all registered **AllChange** users, see the Send Mail to **AllChange** Users chapter of the **AllChange** Reference Manual.

#### Archiving

It may be useful to archive CRs, Parts or Baselines off-line when they become very old and out of date. To this end facilities are provided to the administrator in ACE to perform this function. See the Archiving sections of the Creating and Managing Parts, Baselines and CRs chapters in the **All-Change** User Manual. By default this facility is disabled, it may be enabled by **enabling** the [Archiving feature](#) in the configuration options, this will enable the `archfunc.ac` function definition file.

#### Importing from CSV File

It may be useful to import CR and/or part data from a third party application. To this end facilities are provided to the administrator in ACE to import parts and CRs from a comma separated text file (.csv file). See Importing CRs from CSV File and Importing Parts from CSV File in the reference manual for details. By default this facility is disabled, it may be enabled by **enabling** the [CSV Import feature](#) in the configuration options, this will enable the `csvfunc.ac` function definition file.

#### Import from Subversion

This allows parts to be imported from a subversion repository including the version history. See

Import Parts from Subversion in the reference manual for details. By default this facility is disabled, it may be enabled by enabling the [Subversion Import feature](#) in the configuration options, this will enable the `thesvnfunc.ac` function definition file.

### Check Baseline Check-outs

This function available from the **Baseline** menu will check that the versions checked out are the versions in a baseline, see the Check Baseline Check-outs chapter of the **AllChange** Reference Manual.

### Check Check-out Changes

This function available from the **Workspace** menu will check which workfiles for parts checked out to the current workspace have been changed, see the Check Check-out Changes chapter of the **AllChange** Reference Manual.

### Delete Check-out

This command, available from the **Workspace** menu, allows check-out records to be deleted directly bypassing the normal **Return/Check In** mechanism, see [Delete Check-out](#) chapter of the AllChange Reference Manual.

### Issue

The underlying **Issue** command used by **Check Out** is available from the **Part** menu, see Issue chapter of the **AllChange** Reference Manual.

### Return

The underlying **Return** command used by **Check In** is available from the **Part** menu, see Return chapter of the **AllChange** Reference Manual.

### Rename Part

The Rename Part command may be used to rename a component but without changing any references to it in baselines, see [Rename Part](#) chapter of the **AllChange** Reference Manual. Note that rename does not require a CR even if the class of part has the Require CR attribute. This should only be used when absolutely necessary, in the normal case [Move Part](#) should be used.

### Move Field

This function is available from **Misc | Administrator Tools** and allows data to be moved from one arbitrary field to another. This is useful when an arbitrary field definition is changed from one field to another to move any existing data. See Move Field chapter of the **AllChange** Reference Manual.

### Clear Field

This function is available from **Misc | Administrator Tools** and allows arbitrary field value to be set to blank. This is useful when an arbitrary field definition is removed. See Clear Field chapter of the **AllChange** Reference Manual.

### Fix Field Values

This function is available from **Misc | Administrator Tools** and allows arbitrary field value to be checked for validity. This is useful when an arbitrary field list of values changes. See Fix Field Values chapter of the **AllChange** Reference Manual.

### Set No Keywords Substitution

This function is available from **Misc | Administrator Tools** and descends the Parts Tree explicitly setting the "no keyword substitutions" flag on VC files matching a pattern. This is useful when a binary file has been checked in but its suffix is not in the list of Binary Suffixes and keyword substitutions are not desired on this type of file. See [Set No Keywords Substitution](#) chapter of the **AllChange** Reference Manual.

### VC Command

It may be useful sometimes to access the VC files directly, for example, in order to resolve a discrepancy highlighted by a `pachkvc` report. This function available from the **Misc** menu provides the facilities to do this, see VC Command chapter of the **AllChange** Reference Manual.

### Re-read Project Configuration

This allows certain configuration files to be re-read, this may be useful when the **AllChange**

administrator has made changes to the workspace, roles, pools, access control rules etc and these need to be effected without restarting **AllChange**. This function is classified as dangerous. See Re-Read Configuration Files chapter of the **AllChange** Reference Manual

### Edit Part Read Permissions

This allows the permission to view/read different parts of the parts tree to be specified/edited, see Edit Part Read Permissions in the reference manual.

In addition there are some **General Info** reports which may be useful to administrators, these are:

### AllChange Project Configuration

this will produce a report which outlines your **AllChange** configuration for a project, i.e. the life-cycles, classes etc. that have been defined. This is available from **Report | General Info** on the **Format File** list (file `giconfig.acr`), and as a Word wizard report (**File | New** and select **AllChange Wiz**) in a slightly different variation.

### General AllChange Statistics

this will produce a textual report of various database statistics. This is available from **Report | General Info** on the **Format File** list (file `gistats.acr`)

### Part Location

this will produce a textual report of listing all the parts which have an explicit location field set. This is available from **Report | General Info** on the **Format File** list (file `palocation.acr`)

## Debugging

When configuring the system to site requirements — particularly when writing ACCELcode in the commands, cycles and user-defined function files — there may be times when it would be useful to see exactly what is happening. **AllChange** provides a variety of methods which can be used to aid debugging, including the following:

- Make use of the debug window facility with the **AllChange Debug Window** (ACDBGWIN) program. This is an administrator only program which should be launched before starting ACE in order for it to come into effect. ACDBGWIN, is supplied for use with ACE. It provides an independent scrolling window to which any trace output is sent; unlike the Command Output window this appears immediately and does not receive any other output.
- Switch on the `trace` system flag (available from **Misc | Set System Flag...** in ACE). Trace information will appear in the Command Output and Debug window, interspersed with any other output from commands.

Trace lines start with `Trace >>`, followed by information on what is happening. The following activities produce trace output:

- entry conditions in the commands file
- actions in the commands file
- entry conditions and actions in the cycles file
- exit conditions and actions in the cycles file
- calls to the ACCEL function `interpret()`
- The `trace()` ACCEL function may be used to monitor calls/ assignments to selected ACCEL user-defined functions/variables.
- Switch on the `echo` system flag (available from **Misc | Set System Flag...** in ACE). Certain AllChange operations are echoed to the Command Output window prior to execution, including all command lines generated by ACE, all lines read from a script file of AllChange commands, and any operating system command lines starting with a `@` issued from ACCEL. If this proves too much try switching on just the `echocommand` system flag which echoes just command lines generated by ace
- Select the **Debug mode of startup** ACE option (**Misc | Options** in ACE) and then when **AllChange** starts the `trace` and `echo` system flags will be set.

- Place `trace_echo()` functions in ACCEL code to insert any message into the trace output . The output will appear in the Command Output and Debug windows.
- Place `prompt()` functions in ACCEL code. A dialog displaying the message will appear; execution will not continue until the dialog is dismissed.

## URL Protocol

**AllChange** includes a handler for URLs which allows **AllChange** items to be opened by means of a hyper-link. This link may be used in emails, web pages, Windows shortcuts, etc.

The **AllChange** URL is in the form:

```
allchange://<project>?<itemurl>
```

where:

<project> is the name of the AllChange project

<itemurl> is:

```
[a=<action>&]t=<type>&i=<item>[&nw=<>true|false>][&ws=<workspace>]
```

[] denote optional items

<action> is always `v` (or `view`) or `cv` (or `castvote`) (and will default to `v` if not found)

<type> is one of `c` (or `cr`), `p` (or `part`), `b` (or `baseline`), `is` (or `issuefor check-outs`), `in` (or `instance`)

<item> is the item to be acted on, it may be a part path, a cr number, a baseline name or an instance name. (If viewing an issue/check-out, then <workspace> specifies the workspace and item specifies the part.)

`nw` or `newwindow` may optionally be specified as `true` to force an item to view in a new window.

`ws` or `workspace` specifies the workspace if the item viewed is a check-out record

The <itemurl> and the <project> must have unsafe characters escaped (e.g. space must be `%20` etc).

The URL for viewing an item is included in assignment and vote mailing in the out of the box configurations by using the **GenerateURL** ACCEL function defined in `utility.ac`.

A function, **Generate URL for selection**, for generating a correctly formatted URL from the currently viewed item, or the selected item in a list, is available from context menus of most lists and also to the **Misc** menu. The function will generate the URL and output it to the output window, from where it can be copied to the clipboard for pasting into other applications, or creating a desktop shortcut etc.

(Note that if a project is accessed by alternative names (e.g. when using different interfaces), then the code that generates the URL may be modified to use the correct project name. See the **MakeAC-InterfaceURL** and **MakeWebInterfaceURL** functions in `utility.ac` for details on how to do this.)

The URL is initially processed by the handler, which is provided by an **AllChange** executable named **acvfs.exe**, to remove the protocol prefix, and connect to an **AllChange** with the correct project. The <itemurl> (i.e. the portion of the URL after the `?`) is passed to the running **AllChange** by calling a function called **ProcessURL**, found in `utility.ac`.

When invoking a URL, the handler (`acvfs.exe`) will parse the URL to determine the required project. It will then use OLE to connect to a running **AllChange**. If no **AllChange** is running, or the running **AllChange** is on the wrong project, then `acvfs` runs an ACE with the required project specified as `-EACPROJECT`, and then re-tries connecting. As OLE will always connect to the first running instance it finds, it is not possible to force connection to the ACE just run (if there was already an ACE running), and so a 'prog id' should be specified in the user's profile in order to allow this. This should be in the key `HKEY_CURRENT_USER\Software\Intasoft\AllChange\<version>\<default project>\ACVFS` and should be a string value named `ACEOLESERVER`. The value must be one of the valid registered **AllChange** application IDs, i.e. "AllChange.Application\_N" where N is a number from 1 to 9.

The protocol handler may be registered by using the following command line:

```
acvfs [/s] /register
```

and unregistered using the following:

```
acvfs [/s] /unregister
```

The option '/s' argument specifies silent mode, and prevent acvfs displaying a message box on completion of registering or unregistering.

Registering is carried out during a workstation install, and unregistering during the workstation uninstall.

# Creating and Modifying Report Formats

## About Creating and Modifying Report Formats

The content and layout of reports is defined in external format files. A number of report format files are supplied with the **AllChange** system. These accommodate a wide variety of different types of report such as:

- Textual reports
- Textual Metrics
- Graphical display of Metrics
- Reports exporting data to external applications such as MS Word and MS Excel
- Reports available directly from within MS Word via an **AllChange** Wizard.

The textual reports may be produced using the ACREPORT GUI based reporting tool, this allows prettily formatted graphical reports to be produced using a simple drag and drop interface to define the required layout. Acreport format files have a `.acr` suffix This is the recommended method for creating report formats.

All the above reports may also be produced using the textual report generation facility. Textual reports produced using this are plain ASCII text and may be output to a file as well as the Output Window or a printer. The graphical and external application reports are achieved by exporting the textual results via DDE or OLE to a third party application such as a graph drawer or MS Word.

If you want to alter one of these — or if you want to create a new one from scratch — you will need to understand how report format files are organised, which is described in this section. You will also need to read about ACCEL— see [ACCEL — AllChange Command Evaluation Language](#).

If you create a new report, or if you wish to alter an existing report you should do this using **Plan | Report Formats**, see [Report Formats](#).

Note that all supplied report format files call user-defined functions `FormatDate` and `SysDate` whenever a date is to be output. These functions are defined in the supplied `reportfunc.ac` file. To change the date/ time format used in all reports it is only necessary to alter these functions; the report formats themselves do not need changing. They can be altered if required to use a more user-friendly date or time format such as `windate` (the Windows date format). Beware, however, that the output of `windate` is defined on an individual machine basis so that if this format is used dates in reports may appear in different formats when produced on different PCs. It may therefore be better to define a consistent, company-wide, date format.

Format files should have a `.rep` suffix. They are text files and are edited using ACEDIT, see [The All-Change Editor](#). They may be in the project or system directory .

## Adding New Formats to ACE

Report formats are used by the ACE**Report** dialogs, the ACE **Report Wizard** dialogs and the **AllChange** Word Wizard.

The **Format List** dialog from the **Report** dialogs will show all formats with a filename of the form `<db-prefix><name>.rep`.

The *db-prefix* should be the 2 letter database prefix for the database being reported on.

The report formats available in the **Wizard** dialogs are those with a category of **Brief**, **Detail**, **Graph**, **Status Log** or **Word/ Excel**.

The report formats available from the Word Wizard are those with a category of **Word Wizard**.

In order to add a new report format the **ACCONFIGReport Formats** window should be used, selecting the appropriate category and file name depending on where the report is to be used.

In addition new tabular reports may be added using the column report wizard, see below.

## Columnar Reports in Excel/Word

Report which are to generate columnar (i.e. like a table) output to Microsoft Word or Excel may be created using the **Column Report Wizard** on the **Info** menu in ACE.

Since the report is exported directly to Word or Excel the data can then be sorted on any column and/ or printed using the features of Excel/ Word.

The kind of report that can be produced by this method is restricted to tabular data so that it can be sorted; this precludes outputting sub-lists of data, such as the parts affected for each CR. The ACCEL code could be modified to generate reports which did not have this restriction if sorting were not required. It could also be changed to export to programs other than Excel or Word.

See the **AllChange** User Manual for details of the Wizard dialogs.

The wizard is implement in ACCEL code which is in `wizfunc.ac`, and the template for generating the report format file is in `crwizrep.tpl`. A couple of pre-generated report formats, `blhx1c1.rep` and `crmwcl.rep`, have been supplied in the system directory.

## Report Format Syntax

### About Report Format Syntax

Creation of non ASCII textual reports is supported by ACREPORT. It is recommended that this is used for all textual reports if there is no requirement for the output to be ASCII.

The reporting facilities which allow the creation of ASCII text based reports, exports to third party applications such as MS Excel and MS Word, reports for the Word report wizard and graphical reports require that a report format file is created to define the report. This is an ASCII text file and may be created and modified using ACCONFIG and the **AllChange** text editor (ACEDIT). This also requires a knowledge of ACCEL .

Each report format must define a main database that is to be searched and reported on; the criteria specified will determine which items from the database are included in the report.

A report format may also specify any subsidiary databases to be searched and reported on. In general, for each item in the main database reported on, the next level subsidiary database will be searched and each item matching the criteria for that database will be reported on; this level in turn will search and report on the next level subsidiary database and so on.

The format files consist of four sections, the first three defining the content of the report and the last defining exactly what information to include and how to lay this out.

The start of each section is identified by a keyword which must appear in column 0 and, except for the `IMAGE` section, continues until the first occurrence of a blank line.

The `CONTROL` and `IMAGE` sections are compulsory, the `BREAK` and `DISPLAY` sections are optional.

The section keywords are:

```
CONTROL
BREAK
DISPLAY
IMAGE
```

Lines starting with # (hash) in column 0 are ignored and may be used for comments.

### Control Section

The control section defines what databases are reported on and (together with the command line) which items from that database are included. Control sections are akin to *loops* in ACCEL. An ACCEL `each` loop can be used to visit database records programmatically without producing a report; it uses the same database names, items and arguments as a control section.

The section is of the form:

CONTROL

```

    levelno database-name [items]
    [-all|-edit|-def|-reg|-top] [-uses] [-recursive]
    [-latest how-many]
    [-recurseup] [-progress]
    [-indexedby index [-itemaffectedtype type [-solved]]] [-cond con-
    dition]
  
```

At least one entry of the form shown above must be present, with a *levelno* of 0. This defines the main database to be searched and reported on. Further entries can be used to specify subsidiary databases searched through once for each item included from the previous level.

Each entry consists of a definition of the database used, followed by the specification of any arguments that are to be defined.

Each entry defines one level of reporting.

The *levelno* represents the level of reporting. This should be a number from 0 upwards, with 0 defining the main database to be reported on.

The *database-name* specifies the name of the database to search.

Note that with the use of [nomenclature mapping](#) databases may appear with different naming to the underlying **AllChange** database name in ACE; for example the out-of-the-box configuration maps issue to check-out, thus the issue browser/menu items etc will show as **Check-out**. The underlying **AllChange** database, however, remains issue.

The database name may be one of:

Main Data-bases	Items Affected Data-bases	Status Log Data-bases	Configuration Data	No Data-base
part version issue instance cr baseline baseline_ detail part_vote cr_vote baseline_ vote monitor	part_items_ affected cr_items_ affected baseline_items_ affected	part_status_ log cr_status_log baseline_ status_log	branch class command config cycle cycle_ status field_name field_tab group include partperm pool report_for- mat role username vote_def- inition workspace	none

*Items* specifies which item(s) from the database to report on; it is parsed as an ACCEL expression. *Items* may specify \* to match wildcards, e.g. starts-with\*, \*ends-with, \*contains1\*contains2\* For subsidiary databases, the *items* may be used to define the link between the database and the next level database. In most cases the *items* for a subsidiary database will be a field reference from the previous level's database for which there is a matching field in this level's database. For example, if the main database is **part**, and the subsidiary at level one is **issue**, then the *items* for the subsidiary may be `pa_part` since both the parts database and the issues database have a partname field. This will produce output on any issues of each

part reported on. *Items* may also be a field reference (or function or literal string) which returns a list (or space-separated string) of items. For example, if the main database is **CR**, and the subsidiary at level one is **baseline**, then the *items* for the subsidiary may be `cr_blinesaffected` since both the CRs database and the baselines database have a baseline name field. This will produce output on each baseline affected by each CR reported on. If *items* is omitted *all* items in the database are reported on. On the other hand, if *items* is specified but resolves to the empty string (e.g. there are no baselines affected by a particular CR), or if the specified item does not appear in the database (e.g. there are no issues of a particular part), no output is produced (for this control level).

The `-indexedby` argument may be used to cause a control level to visit records in a user-defined order instead of the default order; this may be used for sorting or efficiency purposes. *Index* identifies the names of fields to filter by joined by /, e.g. `assignee/number` will filter/sort by assignee followed by number. Note that this does not refer to underlying SQL Server indices.

*Note that if using the `itemaffected` field for the `items` affected databases, the `-itemaffectedtype` argument must also be specified. If the versions solved relationship for the part `itemaffectedtype` is required then the `-solved` argument must be used.*

Whenever a report uses `-indexedby`, any items specified on the command line must be of the same type as the first field to be indexed by. So, if iterating through the CRs database indexed by assignee, any items passed on the command line would name assignees, not CR numbers. Thus if no items were specified on the command line the whole of the CRs database would be reported on in CR assignee order; if `fred` were specified on the command line those CRs assigned to `fred` would be reported on.

Databases, items and indexing are discussed in detail in [Databases](#).

Normally, when reporting on databases, a progress bar is displayed as the outer database (control level 0) is traversed. Only one progress bar at a time can be active. A new request for a progress bar is ignored if one is already active. If less than 3 items are to be visited no progress bar is used. In a report format file with multiple control levels, it is possible to change which database displays a progress bar by placing a `-progress` against the desired control level. An ACCEL *each* loop must specify `-progress` explicitly if a progress bar is desired.

The other arguments for each level have the same effect as the corresponding command line arguments and may be used to define search criteria for the report; they are entirely optional. Which arguments make sense with which databases, and what their effects are, is detailed below in [Databases](#). The *condition* may be used to filter out any unwanted items. Any command line arguments specified on invocation of **report** will override any arguments specified in the format file for the main database.

The control section entries tie up with `-CONTROLLEVEL` commands in the image section of the report via the *levelno* (see [Image Section](#)).

### Break Section

The break section defines special break levels that are only included in a report if some condition is true at that point in the report. This section is entirely optional and need not be considered for simple reports. It should be used to put in sections of a report that are only applicable to items that fulfil some specifiable condition. Break sections are akin to *ifs* in ACCEL.

This section is of the form:

```
BREAK
      levelno
      -cond condition
      :
```

Entries in the break level section tie up with `-BREAKLEVEL` commands in the image section of the report via the *levelno* (see [Image Section](#)).

## Display Section

This optional section defines the output filename and page length. These may both be overridden from the command line. If present, this section may include one or more of the lines shown:

```
DISPLAY
    page_length length
    output filename
    append
```

## Image Section

### About Image Section

The image section of the format file defines the actual layout and content of the report.

The image section has the form:

```
IMAGE
    |
    EXIT
```

Within the image section there are three types of entry:

- control lines: identified by a - (minus) character in column zero.
- picture lines: identified by a + (plus) or ! (exclamation) character in column zero.
- field definition lines: all lines not containing a -, + or a ! character in column zero are field definition lines.

### Control Lines

Control lines define the start of a section of the report image. They are denoted by a - character followed by a keyword from the following list. Each section continues until a -END (or -BODY) control line or the start of a header or footer section is encountered.

- REPORTHEADER** denotes the start of the report header section. This section will be reproduced only once in the report at the start of the output (after any page header). It should be used to produce any title or general information relevant to the report as a whole.
- REPORTFOOTER** denotes the start of the report footer section. This section will be reproduced only once in the report at the end of the output. It can be used to produce any information gathered in the course of the report.
- PAGEHEADER** denotes the start of the page header section. This section will be reproduced at the start of each page in the report.
- PAGEFOOTER** denotes the start of the page footer section. This section will be reproduced at the end of each page in the report. It might be used to provide such things as page numbers at the foot of each page.
- BODY** is synonymous with -**CONTROLLEVEL** 0. Each report must have a **BODY** section defining the output to be produced for each item being reported on. This section will also include any sublevels of reporting (defined by the -**CONTROLLEVEL** keyword).
- CONTROLLEVEL** denotes the start of a control level and should be followed by a number, indicating the control level under consideration. Control level zero represents the main body of the report and is synonymous with -**BODY**.

The control levels tie up with the levels defined in the Control Section of the report format. They define the content and layout of the report for each database to be reported on. For each level in the Control Section there should be a corresponding -**CONTROLLEVEL** in the image section.

**-BREAKLEVEL** denotes the start of a break level and should be followed by a number, indicating the break level under consideration.

The break levels tie up with the levels defined in the Break Section of the report format. They define conditional sections of the report which will be included only for those items matching the condition given in the Break Section. For each level in the Break Section there should be a corresponding **-BREAKLEVEL** in the image section.

**-END** signifies the end of a section.

**-PAGETHROW** is the only form of control line that does not signify the start or end of a reporting section.

This keyword may be followed by an optional number. If the number is absent then a new page will be thrown whenever this point is reached in the output. If a number is specified then this indicates the minimum number of lines that must remain available on the page for output to continue without a page throw; if less than this number are available then a new page will be thrown.

**-CONTROLLEVEL** and **-BREAKLEVEL** sections may be placed inside **-REPORTHEADER** and **-REPORT-FOOTER** sections (and inside **-PAGEHEADER** and **-PAGEFOOTER**, though this may not be useful).

## Picture Lines

There are two types of picture line:

1. Output picture line - this will produce output in the report. An Output picture line is signified by a + (plus) character at the start of the line
2. Expression picture line - this will produce no output but allow arbitrary ACCELexpressions to be evaluated. An Expression picture line is signified by a ! (exclamation) character at the start of the line

Each output picture line (signified by the initial + (plus) character) represents a line of output in the report. An output picture line can be made up of:

- literal text, which will be output directly,
- fields, which will have values substituted into them, as specified on the following field definition line(s). A field is indicated by an @ (at) character, followed by zero or more other characters indicating various attributes of the field;
- Word wizard directives. These will cause formatting commands to be evaluated for reports invoked from the Word Wizard. Wizard directives have the form *\$directive\$*.

A literal @ (rather than a field start) may be embedded in a picture line by preceding it with a \ (backslash).

The total width of the field is (normally) indicated by the total number of field characters (including the @, any field attribute qualifiers, and the string of field attribute characters). If the text of the substitution is shorter than the field width it will be padded with space characters to the field width. If it is longer it will usually be truncated with a \* (asterisk) indicating the point of truncation (though it may be split onto subsequent lines). The field ends when the first character which cannot be part of the field (including another @ character) is encountered.

The field attributes may be a string of any one of the following characters:

- < (less-than) indicates that the text of the field should be left justified, i.e. any blank padding should happen on the right of the field.
- > (greater-than) indicates that the text of the field should be right justified, i.e. any blank padding should happen on the left of the field.
- # (hash) indicates that this is a numeric field and should be right justified. This has the same effect as the > attribute.
- | (bar) indicates that the text should be centred within the field.
- } (close-brace) indicates that no padding, truncation or splitting should take place. The text will take up only as much space as needed, no more and no less. This is the only time that the width of the field within the picture line is ignored. (There is no point in having more than one } character since the field width is ignored.)

The field attribute string may optionally be preceded by any of the following field attribute modifier characters:

- \* (asterisk) indicates that, if truncation is necessary, the field will be truncated on the left, rather than the right.
- ^ (circumflex) indicates that, if the text of the field is longer than the field width, it should spill onto subsequent lines beneath the field rather than be truncated to fit.
- ~ (tilde) indicates that a multiple item field should be tabularised across several lines rather than being output on a single line as is normally the case. The next character in the field should be a number, indicating the number of columns to use, or a ? (query), indicating that as many columns as possible fitting on a line should be used. (If neither is present then the items will be output on one line, separated by space characters.) Hence, for example, using 1 as the number results in one item being output per line.

ACCEL functions returning a list are examples of multiple item fields for which this functionality may be useful.

The following directives may be used in picture lines: they will be recognised by the **AllChange** Word wizard and converted into suitable Word formatting statements.

The following directives are supported within a picture line in a report format file:

```

$STARTBOLD$ ... $EN- Formats the intervening text as bold
DBOLD$

$STARTITALIC$ ... Formats the intervening text as italic
$ENDITALIC$

$STARTUNDERLINE$ ... Underlines the intervening text
$ENDUNDERLINE$

$STARTFONT=font$ ... Formats the text in the specified font
$ENDFONT$

$STARTSIZE=size$ ... Formats the text in the specified size
$ENDSIZE$

$STARTCENTRE$ ... Centres the intervening text
$ENDCENTRE$

$STARTRIGHT$ ... Right justifies the intervening text
$ENDRIGHT$

$STARTLEFT$ ... Left justifies the intervening text
$ENDLEFT$

$STARTCOLOUR=colour$ ... $ENDCOLOUR$ Formats the text in the specified colour. colour may be one of: Auto, Black,
Blue, BrightGreen, DarkBlue, DarkRed, DarkYellow, Gray25, Gray50, Green,
NoHighlight, Pink, Red, Teal, Turquoise, Violet, White and Yellow.

$STARTUNDERLINE=style$ ... $EN- Formats the text in the specified underline style. style may be one of: None,
DUNDERLINE$ Dash, DotDash, DotDotDash, Thick, Dotted, Double, Single, Words and
Wavy.

$STARTSTYLE=<Style Formats the intervening text with the name of the specified Word style
Name>$ ... $EN-
DSTYLE$

$STARTSMALLCAPS$ ... Formats the intervening text to be small caps.
$ENDSMALLCAPS$

$STARTSTRIKETHROUGH$ ... $EN- Formats the intervening text to be struck through.
DSTRIKETHROUGH$

$STARTHEADERtext$EN- Places the text in the word document header. This may only occur once per
DHEADER$ report.

$STARTFOOTERtext$EN- Places the text in the word document footer. This may only occur once per

```

DFOOTER\$ **report.**

The following directives should be placed, one per line, on their own picture lines around the required block of text. The line(s) containing the instruction(s) will be removed from the final output:

```
$STARTFRAME [=height-of-frame] $
|
$ENDFRAME$
```

encloses the text block in a frame; see WordBasic `InsertFrame`, `FormatFrame` statements.

```
$STARTBORDER [=border-line-style] $
|
$ENDBORDER$
```

encloses the text block in a border; see WordBasic `Border...`, `BorderLineStyle` statements.

```
$INSERTLINEABOVE [=line-style] $
$INSERTLINEBELOW [=line-style] $
$INSERTPAGEBREAK$
```

inserts a line or a page break; see WordBasic `Border...`, `BorderLineStyle`, `InsertPageBreak` statements.

```
$STARTTABLE=auto-table-style[,Commas][,NoBorders][,NoShading][,NoFont]
|
[,Colour][,NoAutoFit][,FirstRow][,FirstColumn]
[,LastRow][,LastColumn][,BaseLayout=width]
[NoRBorder][ColWidths=argument] $
|
$ENDTABLE$
```

encloses the text block in a border; see WordBasic `TableAutoFormat`, `TableInsertTable`, `Text-ToTable` statements. The possible arguments to `$STARTTABLE$` are:

<b>auto-table-style</b>	indicates which table style to use
<b>Commas</b>	split on commas (default is tabs)
<b>NoBorders</b>	indicates no borders in table
<b>NoShading</b>	indicates no shading in table
<b>NoFont</b>	indicates no bold/italic formatting
<b>Colour</b>	indicates respect colour info
<b>NoAutoFit</b>	do not auto-fit columns
<b>FirstRow</b>	respect special first row formatting
<b>FirstColumn</b>	respect special first column formatting
<b>LastRow</b>	respect special last row formatting
<b>LastColumn</b>	respect special last column formatting
<b>BaseLayout=width</b>	use predefined layout with odd columns of width specified (default 1")
<b>ColWidths=argument</b>	specify the width of each column of the table; <i>argument</i> should be of the form <i>col1 width   col2 width</i> .... This feature is not supported under Word95.
<b>NoRBorder</b>	If specified all columns in the table will have equal width vertical borders. The default case (i.e. <code>NoRBorder</code> is not specified) is for the right vertical border of odd columns to be thinner than those of the even columns. In this way pairs of odd and even columns appear to be grouped which is often desired if the odd columns is a label and the corresponding even column the actual value.

It is also possible for expressions to be evaluated in the image section without producing any output. If a picture line starts with a ! (exclamation) instead of the usual + no output will be produced for that line; the

rest of the line is ignored. The following lines — which would normally be field definition lines for the fields in the picture line — may contain any desired ACCEL expressions. These are evaluated when the preceding picture line is encountered as usual, but no line will be output.

This facility may typically be used to increment counters so that statistics about the items found by the report can be output at the end. A template to count the number of items found in the course of a report might be something like:

```
IMAGE
-REPORTHEADER
! (Next line is evaluated)
setvar(count, 0)
-BODY
! (Next line is evaluated)
setvar(count, var(count)+1)
-REPORTFOOTER
+ Total = @<<<<
var(count)
-EXIT
```

### Field Definition Lines

Any picture line containing one or more fields to be filled in must be followed by a field definition line(s) defining the textual value to be inserted into each field. Each field definition may be any valid ACCEL expression (see [ACCEL — AllChange Command Evaluation Language](#)), the output from which will be inserted into the picture line.

### Databases

#### About Databases

This section describes what kind of items a report on a database accepts on the command line *assuming the database is being read in the default indexing order*. It states what the default index order for each database is. It also explains the effects of any arguments which depend on the database. If the `-indexedby` argument is used to visit records in a user-defined indexing order any items specified must be of the same kind as the index, as described in [Control Section](#) and Indexing Databases.

Note that with the use of [nomenclature mapping](#) databases may appear with different naming to the underlying **AllChange** database name in ACE; for example the out-of-the-box configuration maps issue to check-out, thus the issue browser/menu items etc will show as **Check-out**. The underlying **AllChange** database, however, remains issue.

### Baselines

The default index is by baseline name. Baseline reports normally just include information from the baseline header. If the `-all` argument is specified then the body of the report will be produced once for each baseline detail.

The `-latest` argument may be used to limit the number of versions of each baseline reported on to the number specified (e.g. `-latest 2` will limit the baseline versions reported to the latest 2 versions).

The `-recursive` argument will cause all of the subbaselines of a meta-baseline to be visited (recursively) in turn.

If particular baseline names are specified then the report will be restricted to those baselines. If no baselines are specified then all baselines will be included in the report. A range of baseline names is accepted.

## Baseline Details

Although the distinction is mostly hidden, there are actually two databases for baselines: one for the headers (`baseline`) and one for the details (`baseline_detail`). Although baseline details may be visited/ reported on by using the `-all` argument on the `baseline` database, this is actually going through the baseline headers database and then visiting each baseline's details. Specifying `baseline_detail` for the database goes through the baseline details database instead. This is important if it is desired to visit the records in non-default index order.

The corresponding baseline header is *not* read by default (for speed), so in such a loop ACCEL field references to a field which lives in the baseline header but not in the baseline detail (e.g. `bl_comment`) are undefined. If the `-all` argument is used the corresponding baseline record is read, so these field references may then be used; the algorithm only reads the baseline when it has changed from the previous baseline detail read, so this will be efficient.

The default index is by baseline name, like the baseline headers database. If particular baseline names are specified then the report will be restricted to those baselines' details. If no baselines are specified then all baselines' details will be included in the report.

As a special case, when iterating through the baseline detail database indexed by partname the `-recursive` argument can be used in combination with a component having a trailing `;` to visit the baseline details of all versions of the component. A `-recurseup` argument is also recognised: if the baseline in which a detail appears is itself a detail in a meta-baseline it follows such baseline details recursively *up* visiting each in turn; in other words, starting from a partname or baseline name it visits every (meta-)baseline in which it appears.

## Change Requests

The default index is by CR number. The numbers of those Change Requests of interest may be specified on the command line (full CR numbers must be specified, e.g. `CR00017` may not be abbreviated to `CR17`). If no change request numbers are given then all change requests will be included. A range of CR numbers is accepted.

## Issues

This is the database which logs all check-outs.

The default index is by full version. If any partnames are supplied on the command line then all check-outs of those parts will be included in the report. If no partnames are given then all check-outs will be included in the report. A range of partnames is accepted.

If the `-recursive` argument is specified then issues of all children of the specified parts will be included in the report.

## Instances

The default index is by full instance path. The `-recursive` argument will cause all instances in the part tree below the part specified to be reported on.

## items affected

The relationships to parts, crs and baselines from the other databases are reported on separately as though they were different databases. The database names are `part_items_affected`, `cr_items_affected` and `baseline_items_affected`. The default index order is by itemaffecting and the Items specified on the command line will be taken as partnames, baseline names and CR numbers respectively.

If the index is the item affected index then the item affected type must also be specified using the `-item-affectedtype` argument. If solved relationships are needed for part items affected then the `-solved` argument must also be specified.

## Monitors

The default index is by item. 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.

## Parts

The default index is by full partname. A report on the parts database will include those parts named on the command line (or in the format file). Normally the body of the report will be reproduced once for each named part. Specifying `-all` will add an entry for each version of the named parts. Similarly, the `-def`, `-reg`, `-top` and `-edit` options include the default, registered, top and edit versions respectively. Specifying `-recursive` will cause all the children of each named part to be described. Specifying `-uses` will cause any uses-type parts to be followed.

## Status Logs

Each of the different kinds of items whose status changes may be logged — parts, baselines and CRs — are reported on separately as though they were really different databases. The database names are `part_status_log`, `baseline_status_log` and `cr_status_log`; items specified on the command line will be taken as partnames, baseline names and CR numbers respectively. As a special case, when iterating through the part status log the `-recursive` argument can be used in combination with a component having a trailing `;` to visit the status log of all versions of the component. The default index is by item. If no items are specified then all logs of the matching kind will be included in the report.

## Versions

Although the distinction is mostly hidden, there are actually two databases for parts: one for components and subsystems (`part`) and one for the versions of each component (`version`). Although versions may be visited/reported on by using the `-all` argument on the `part` database, this is actually going through the parts database and then visiting each component's details. Specifying `version` for the database goes through the versions database instead. This is important if it is desired to visit the records in non-default index order.

The corresponding component is *not* read by default (for speed), so in such a loop ACCELfield references to a field which lives in the component but not in the version (e.g. `pa_arb1`) are undefined. If the `-all` argument is used the corresponding component record is read, so these field references may then be used; the algorithm only reads the component when it has changed from the previous version read, so this will be efficient. If the `-recursive` argument is specified then any versions starting with the specified string will be included in the report; specifying a component having a trailing `;` will visit all versions of that component.

The default index is by full version name. If particular versions are specified then the report will be restricted to those versions. If no versions are specified then all versions will be included in the report.

## Votes

Each different type of item that can be voted on - parts, baselines and CRs - are reported on separately as though they were really different databases. The database names are `part_vote`, `baseline_vote` and `cr_vote`. The default index is by item and hence parts, baselines or CRs should be specified on the command line.

**Classes**

The only index is by class name. This will report on all classes whose names are specified on the command line. If no class names are given then all classes will be included in the report.

**Command Definitions**

The only index is by command name. Those command definitions whose command names are specified on the command line will be included in the report. If no commands are named then all command definitions will be included.

**Cycles**

The only index is by cycle name. Without the `-all` argument, information on each cycle named on the command line is provided, the body of the report produced once for each cycle. If no cycle names are given then all cycles defined to the **AllChange** system will be included in the report.

If the `-all` argument is specified then the body of the report is produced once for each status of each specified cycle and information on specific statuses will be made available.

**Field Tabs**

The only index is by database name. If a database name is specified then those field tabs applicable to that database will be included in the report, otherwise all field tabs will be included in the report.

**Pools**

The only index is by pool name. Items specified on the command line will be taken to be the names of pools to be reported on. If no pool names are specified then all pools will be included in the report.

**Roles**

The only index is by role name. This will report on the specified roles. If no roles are specified then all roles defined to the **AllChange** system will be included.

**Vote Definitions**

The only index is by cycle name. If a cycle is specified then those vote definitions for that life-cycle will be included in the report, otherwise all vote definitions will be included in the report.

**Workspaces**

The only index is by workspace name. This will report on all workspaces defined to the **AllChange** system matching those workspace names given on the command line. If no workspace names are specified then all workspaces will be included.

**None**

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 in the `CONTROL` section at the beginning 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 can also be used in `-CONTROLLEVELS` within a report format file when there is a list of items that is to be laid out one per line in a particular way. Usually control levels mark an area of a report format file that is output for each selected item in a database other than the primary database. By defining a con-

control level which has `none` as its database and a list as its items the section is output for each item in the list; the `ACCEL` variable `item` is set to each item in turn.

This makes it possible to write report formats which examine records to gather data in the body loop and then wish to iterate over a list they have built up outputting results in rows (e.g. in the `-BODY` of the report examine each CR storing up a list of every status encountered and how many CRs are in that status, then in the `-REPORTFOOTER` iterate over the statuses outputting the totals).

## Indexing Databases

All databases are accessed by default in the *default index order*. Alternative user-defined index orders may be used for "real" databases (e.g. parts, versions, baselines, baseline details, issues, monitors, CRs and status logs, but not pools, classes, workspaces, roles or cycles).

The indexing order used when running a **report** command, executing a control section from a report format file, or executing an `ACCEL` `each` loop determines the order in which records are visited (and hence appear in a report's output). If no explicit index order is specified the default is used.

If no items are specified all records are visited; the index order used affects only the order in which records are visited. If one or more items are specified then only matching records are visited. It is *immensely* faster to use an index order to find an item than to search all records looking for a field of interest. For example, to report on all CRs assigned to a certain user it is much better to use the CR indexed by assignee and specify the username of interest than it is to go through CRs in the default index order (by CR number) picking out those assigned to the user via a break level. The same would apply if writing an `ACCEL` `each` loop to visit CRs assigned to the user.

When iterating through a database on an explicit index order (i.e. using `-indexedby`), the **report** command supports the specification of both an index value and a "standard" item value. The syntax of the **report** command includes:

```
report [-indexedby field [-indexitem index-item]] [db-item[s]]
```

If `-indexedby` is *not* specified the database is visited in "standard" (or "default") index order (e.g. CR number for the CRs database, baseline name for the baselines database), and if any *db-items* are specified only those items are visited.

If `-indexedby` *is* specified the database is visited in that index order. If `-indexitem` is *not* specified then if any *db-items* are specified these are sought in the specified index order. If `-indexitem` *is* specified then *index-item* is what is sought in the index order. *Index-item* is actually treated as a list of items to be sought: records matching any of the index items are included in the report. If in addition any *db-items* are specified then any records visited are examined to see whether their "main" field (i.e. the field they would normally be indexed on, e.g. CR number for the CRs database etc.) matches the *db-items*. Only the **report** command accepts `-indexitem` and the specification of *db-items* which are not sought in the index order; this is used in `ACE` to allow the user fine control over what items are included in a report and what order they appear in. `ACCEL` `each` loops do not recognise `-indexitem`; they look for any *db-items* in the index; if the "main" field of records is of interest it should be examined inside the `each` loop.

Here are some examples:

```
report ...
```

visits all CRs in standard CR number order

```
report ... CR00001 CR00006 CR00111
```

picks just the specified CRs via the standard CR number order

```
report ... -indexedby assignee
```

visits all CRs in assignee order

```
report ... -indexedby assignee fred
```

```
report ... -indexedby assignee -indexitem fred
```

picks just those CRs assigned to fred via the assignee index order

```
report ... -indexedby assignee fred jim sheila
```





would produce a text file, `outfile.txt`, containing the result of the report. If the current user is `mike` then the contents of `outfile.txt` might look something like:

```
-----
/u1/mike/source/fred.c;004 1997/06/12 11:19:41
Author: mike 1997/06/11 17:13:11
Baseline: all 1997/07/01 09:54:19
Baseline: bline1 1997/11/09 14:23:17
-----
/u1/mike/source/jim.c;004 1997/06/09 15:23:15
Author: mike 1997/06/09 14:33:12
Baseline: all 1997/07/01 09:54:19
Baseline: bline1 1997/11/09 14:23:17
-----
```

In order to include all parts checked out (regardless of to whom they are checked out) the following command line could be used:

```
report -format example -output outfile.txt -cond true
```

The condition given on the command line overrides the one in the format file that restricted the report to check-outs made by the current user.

### Integration with Excel Charts

Two sample report format files, `crxlstat.rep` and `crxltrnd.rep`, are supplied with the system. They demonstrate how statistical information on CRs may be exported to an Excel spreadsheet; the spreadsheets include a chart (graph) on the data. They may serve as a basis for further export/ import of information with Excel. Excel must have been properly installed on the machine.

# Upgrading to a New Version of AllChange

## About Upgrading to a New Version of AllChange

In view of the fact that it may take a while to reconfigure the system to your requirements (depending on how much tailoring of the existing system has been performed) we would strongly recommend that you maintain the current live system unchanged whilst implementing the upgrade in parallel.

The following actions and precautions should be taken in order to achieve this:

1. Save your current workstation environment setup (e.g. program items/ icons). If you need to access the live system you can create icons which point to the live system.
2. Install your new version of **AllChange** from the supplied media to a new location . *We strongly recommend that you install the new version of the software to a different location from the current installation. During the install we recommend you select a different program group for shortcuts for the new version, thus maintaining your live shortcuts in tact.*
3. Create a copy of any project directories in order to perform any reconfiguration in the copy, allowing users to continue using the current live system whilst reconfiguration is undertaken.

*The upgrade tool will assist you with this operation*

4. System wide defaults and template files that have been tailored should be reconfigured as necessary in the new system directory . The only configuration files you should have changed will be the users registration files (`users.ac` and `userinfo.ac`), mail mapping file (`mailmap.ac`), FTP user information if using FTP (`ftpusr.ac`) and report format summary file (`report.sum`). — this should be copied from the current system directory to the new system directory. In addition your project definitions will need to be copied or recreated (`projects.ac`)

*This will be performed for you by the upgrade tool*

Other files may have been changed and should be reconfigured as necessary.

5. If necessary the database should be converted to make it compatible with the new version of **All-Change**.

*The upgrade tool will perform this for you.*

6. Your tailored project specific system files will now need to be reconfigured as necessary. If you are using **AllChange** to version control your configuration files (AC4AC) you should ensure that the latest versions are checked in and baselined prior to reconfiguration. After reconfiguration you should check in and baseline the upgraded configuration files

Any files which are not compatible will be highlighted in the appropriate upgrade notes.

*The upgrade tool provides you with facilities to automate any reconfiguration necessary and to convert incompatible configuration files*

7. Read the **Upgrade Notes** and the **Upgrade Instructions** if there are any.
8. Having performed any reconfiguration and testing necessary in the copy project it can be made live by:

- Ensure no users are using the live system.
- Remove (or rename) the current live system directory hierarchy.
- Rename the new system directory to the old system directory name.
- Copy (or check out from AC4AC) any playground configuration files to the live project directory . Take care with `pools.ac`, `workspcs.ac` and `ftpwspc.ac`: these will probably not want to be copied and any required modifications made directly in the live project directory. *This may be performed using the upgrade tool.*
- Perform any conversions required on configuration files in the live project, e.g. `pools.ac`, `workspcs.ac` and `ftpwspc.ac`. *This may be performed using the upgrade tool.*

- Perform any database conversions necessary on the live database. *This may be performed using the upgrade tool.*
  - Modify the *project definition* if necessary for the new system.
  - Modify all users' icons, registry entries and any local copied of executables if necessary (perform a workstation install).
  - Perform testing.
  - Make the new system available to users.
9. If using AC4AC then you will need to upgrade this first prior to upgrading your Live system.  
*The upgrade tool supports this*
10. If you have Play or Test project as well as your live, then this needs to be upgraded as well using the same configuration files as the upgraded Live system.  
*The upgrade tool provides facilities to do this for you if using AC4AC.*

## The Upgrade Tool -- ACUPGRAD

### About The Upgrade Tool -- ACUPGRAD

The upgrade tool supplied with **AllChange** may be invoked from the installed shortcut **AllChange Upgrade Tool**: *note that this will only be available for administrators*. Alternatively you can create your own shortcut specifying any required command line arguments in the normal way (e.g. `ACPROJECT=`).

The upgrade tool is called ACUPGRAD.

It provides facilities to assist in and guide you through the process of upgrading to a new version of **AllChange**.

*It should be used in conjunction with any Upgrade Instructions supplied with the new version.*

It provides facilities to implement the 3 major stages involved in upgrading an **AllChange** project :

1. Perform any necessary database conversion
2. Upgrade any configuration files
3. perform any post processing required on the database

ACUPGRAD will create a new *upgraded* project from an old project by copying files from the old project directory to a new project directory performing whatever conversion process is necessary for each file in turn and create a copy of the database with any necessary conversions. This new project should be regarded as a *play* or *temporary* project for use whilst the upgrade process takes place and whilst the administrator performs any testing etc necessary before the upgraded project can *go live*.

When the administrator is satisfied that the upgrade is complete, then the converted configuration files must be made *live* and the live database must be processed and converted together with any configuration files which still need to be converted in the *live* project.

ACUPGRAD will assist you with both the creation of the temporary project and the go live process. Furthermore, if you are using **AllChange** to version control your configuration files (AC4AC), ACUPGRAD will integrate with your AC4AC and check in and baseline the upgraded configuration files. ACUPGRAD also supports upgrading additional projects from the same configuration if under AC4AC control, for example Play, Live and Training projects.

### The Upgrade Process

The upgrade process may involve upgrading more than one project if you have Play and Live (and possibly other projects), and also if you are using **AllChange** to version control your Live/Play projects (AC4AC), then this itself needs to be upgraded as well. The upgrade tool, ACUPGRAD, supports all of these requirements.

The upgrade process for an **AllChange** Project is a 3 stage process as follows:

1. The first stage involves converting the database to be compatible with the new version of **All-Change**. Whilst this is called the *first* stage - it may actually be repeated and does not *have* to be performed first.
2. The second stage involves copying and processing each configuration file. This may be performed in a single operation processing each file in turn prompting for input when required, or may be manually invoked for each file from a list. This is likely to be the most time consuming part of the upgrade process and will require the most input from the **AllChange** administrator. This may be performed before or after stage 1.
3. The third stage performs any post processing that may be required on the converted database. This stage may *not* be performed until both stages 1 and 2 have been completed. This stage may also require that the configuration files for the *original* project are available.

ACUPGRAD will require information about the location of various **AllChange** components in order to upgrade a project as follows:

**Old System Directory** : this should specify the **AllChanges** system directory for the **AllChange** version that you are upgrading from. This must contain files as installed from the original media (with the exception of the user registration, mail mapping and project definitions files).

**Old Project**: this should specify the either the project name or the project directory (depending on the version you are upgrading from) for the project that you wish to upgrade.

**New Project Name**: this specifies the name you want given to the new project that ACUPGRAD will create.

**New Project Directory**: this should specify the directory which you wish to use as the new *play* or *temporary* project whilst the upgrade process is taking place.

**New Project Description**: this should specify a description for the new project.

**SQL Server name**: this should specify the name of the SQL server instance to be used.

**Authentication**: this should specify Windows or SQL Server authentication. If SQL Server authentication is used then the **User name** and **Password** must be specified.

**Database name**: this should specify the name of the SQL server database for the new *play* or *temporary* project whilst the upgrade process is taking place.

**Encrypt connection**: this should be checked if the data passed between server and clients is to be encrypted.

**Name of AC4AC project**: this should specify the name of the AC4AC project which controls the configuration files for the project to be upgraded if the project is controlled by AC4AC

Having created a complete and converted *temporary* project you should then perform the following operation to ensure that the temporary project will not affect the original project in any way whilst testing takes place.

1. copy any VC files that you require for testing purposes into the appropriate VC file directory in the temporary project directory.

*The upgrade tool will offer to do this for you after converting the database during Stage 1 - if desired only a subset of the VCFILES may be copied to reduce the space taken by the temporary project. Note however, that if only some VCFILES are copied then only those parts for which their corresponding VCFILES have been copied may be used in full - e.g. for check in/out*

2. Modify any pools, workspaces or ftp workspaces to refer to unique directories for the *temporary* project. Note that the upgrade tool will prompt you to do this.

On starting ACUPGRAD you will be presented with a wizard to guide you through the upgrade process.

The steps necessary to upgrade AllChange where there are Live and Play projects which are under AC4AC control are outlined below:

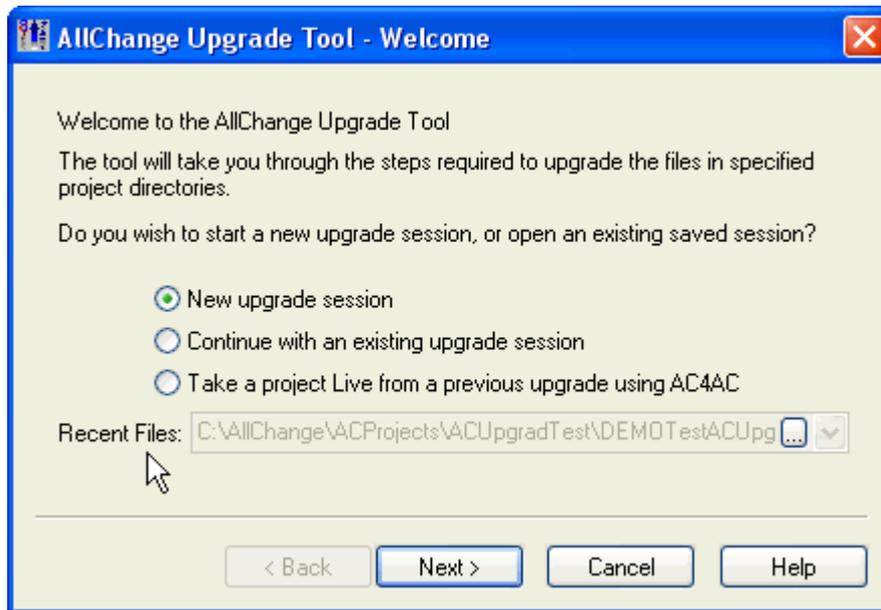
1. Install AllChange to a separate directory, install ACINETD with a different name and port if wishing to test the upgrade client/server and run the current live system at the same time on the same server - see [Running multiple instances of ACINETD](#).
2. Upgrade the AC4AC project to a test AC4AC project stages 1,2 and 3 using the upgrade tool
3. Copy VCFILES to test AC4AC project if not performed by upgrade tool
4. Change workspace definitions in test AC4AC to unique directories
5. Test upgraded test AC4AC project
6. Go Live with AC4AC project using Upgrade tool, stages 1, 2 and 3 - clean up test AC4ACproject if desired.
7. Upgrade Live project to test project using upgrade tool, stages 1,2 and 3. This will check out for edit configuration files under AC4AC control to a workspace defined for the test project
8. Copy (some) VCFILES to test project if not performed by upgrade tool
9. Change workspace/pool definitions in test project to unique directories
10. Test the test project
11. Lock users out of Live project in preparation for going live with the upgrade - ensure no users are logged into the Live project
12. Go Live with the test project using the upgrade tool, this will check the configuration files under AC4AC control into the AC4AC project and add them to a baseline. Perform stages 1,2 and 3. The files under AC4AC control will be checked out read only from AC4AC to the workspace for the Live project
13. Ensure all users locked out of the Play project
14. Go Live with Play project using upgrade tool. Perform stages 1, 2 and 3. This will check out read only the configuration files under AC4AC control to the workspace for the Play project.
15. Remove the ACINETD for the new version - see [Running multiple instances of ACINETD](#). Reinstall it with default name on default port. You can use the Server Install to do this.
16. Delete (or rename) the old system directory to ensure that users cannot access it accidentally.
17. Remove lockouts to allow users access to Live and Play projects
18. Perform workstation install for all users for new system directory

## The Upgrade Wizard

### *About The Upgrade Wizard*

The **Upgrade Wizard** will guide you through the process of upgrading an **AllChange** project to a new version of **AllChange**. It is accessible from the **File** menu and it will be started automatically on entering the ACUPGRAD tool.

The first screen asks you to select whether you wish to continue with an existing upgrade session, start a new upgrade session or go live from a previous upgrade using AC4AC.



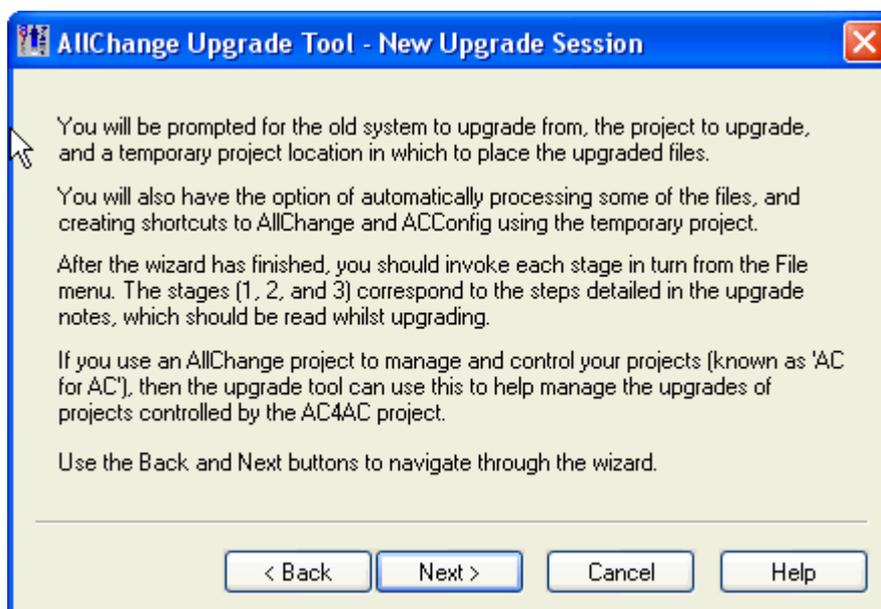
Initially you will start a new session to upgrade a project. This may be a lengthy process and you may not be able to complete the upgrade into the test area in one sitting.

When you wish to continue with an existing session, use this option.

When you have completed the upgrade process and are ready to *go live* with the new version of **All-Change** and the upgraded project, you may use the go live option.

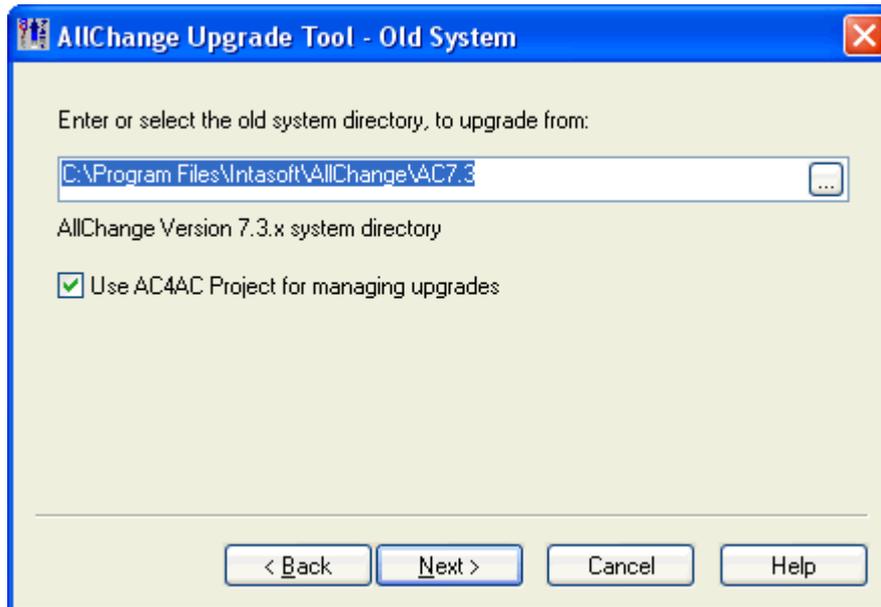
### New Upgrade Session — Information

Having selected to start a new upgrade session the next wizard screen provides you with useful information; read it and when ready select **Next**.



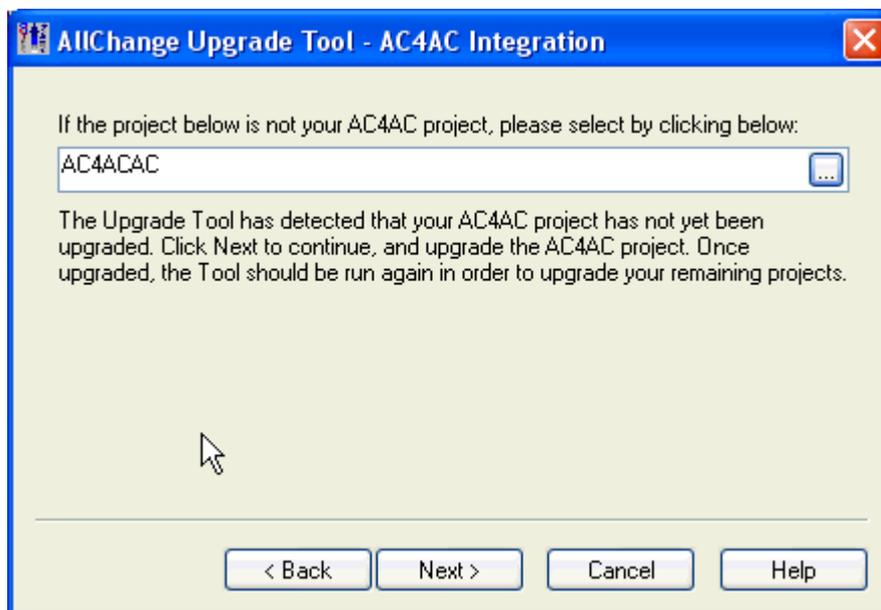
**New upgrade session — Old System**

This screen requires that you specify the system directory for the **AllChange** version that you are upgrading from. If you use AC4AC to manage your configuration files then select this box



**New upgrade session — AC4AC Integration**

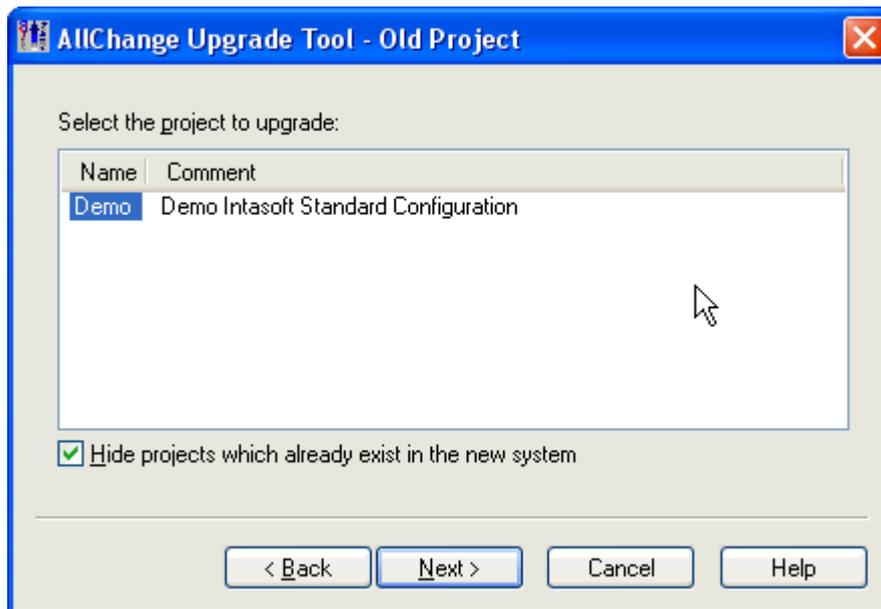
If you selected to Use AC4AC Project for managing upgrades in the previous screen then the next screen will request you to select your AC4AC project. It will then detect whether this has already been upgraded or itself requires upgrading. If it does then it will continue with this as the project to upgrade. Otherwise it will continue to request the project you wish to upgrade.



**New upgrade session — Old Project**

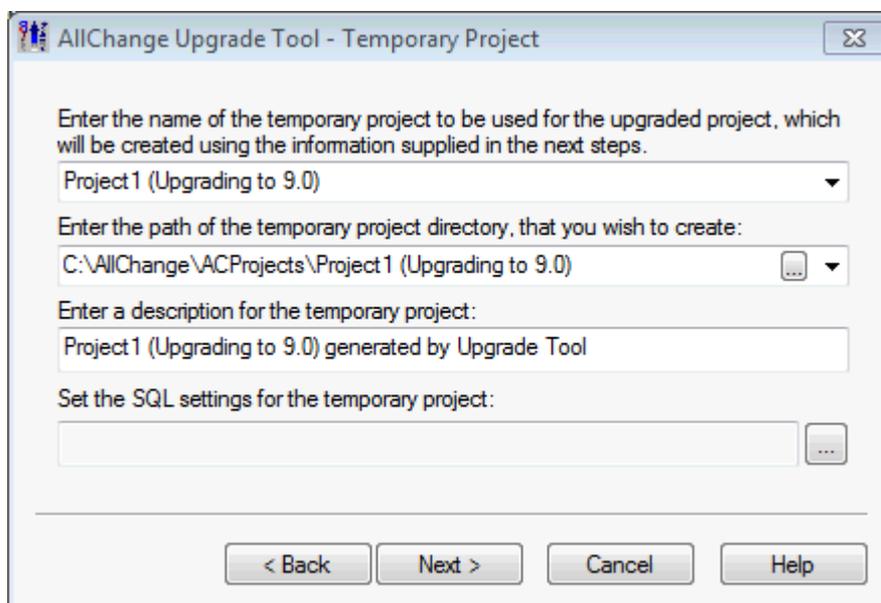
This screen requires that you specify the name of the old project that is to be upgraded.

If **Hide projects which already exist in the new system** is selected, then any projects with the same name defined in the new system directory will be excluded from the list of projects to upgrade.



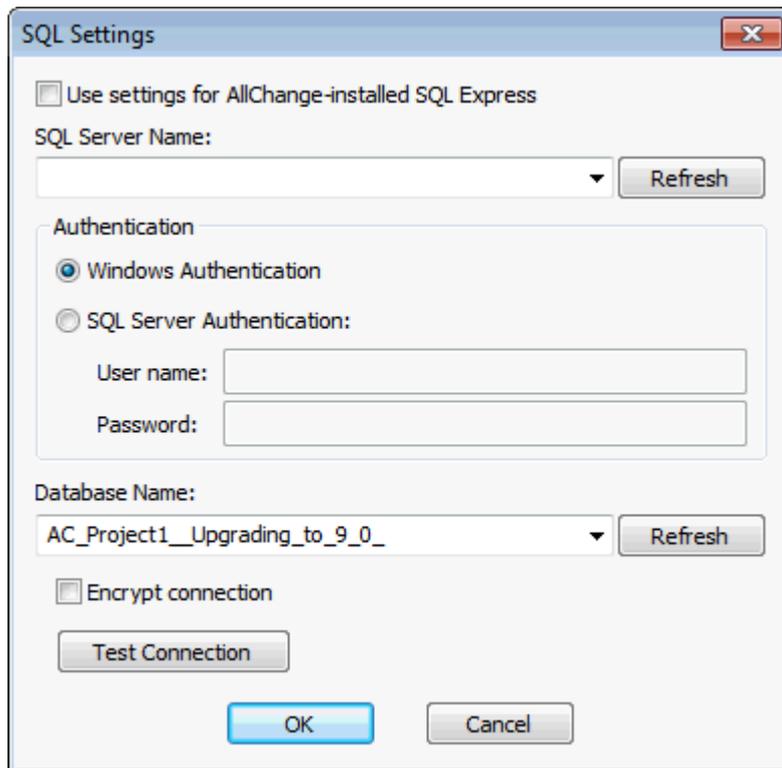
### New upgrade session — New Project

This screen requires that you specify details of the new project to be created as a *playground* or *temporary* project whilst the upgrade takes place.



You must specify a name for the project, the directory which is to be used as the project directory and a description of the project.

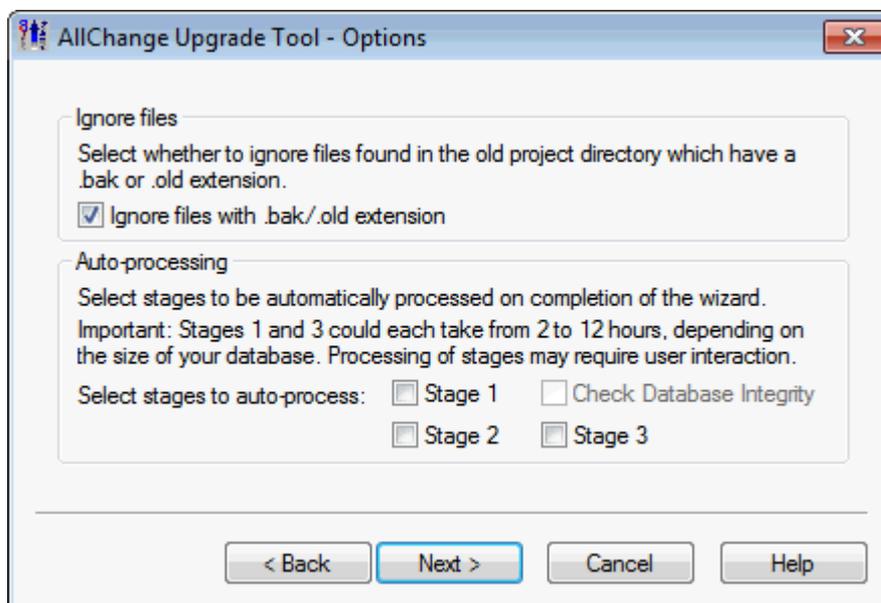
It will also need the settings for SQL server which may be set by clicking the ... button:



see [The Upgrade Process](#) and [Creating and updating AllChange Projects](#). Where possible default values will be entered by the upgrade wizard.

### New upgrade session — Upgrade Options

This screen provides various upgrade options.



The **Ignore files with .bak/.old extension** option should be selected if you do not wish files with this extension which exist in the **Old Project Directory** to be copied to the **New Project Directory**. It is recommended that this is selected.

The **Auto-processing** options allow you to select which stages of the upgrade process you would like to be performed automatically on completion of the wizard.

The stages are:

**Stage 1:** convert the database. If this is selected then you may also select to **Check Database Integrity** prior to conversion of the database. See [Stage 1](#)

**Stage 2:** process all configuration files in turn, see [Stage 2](#)

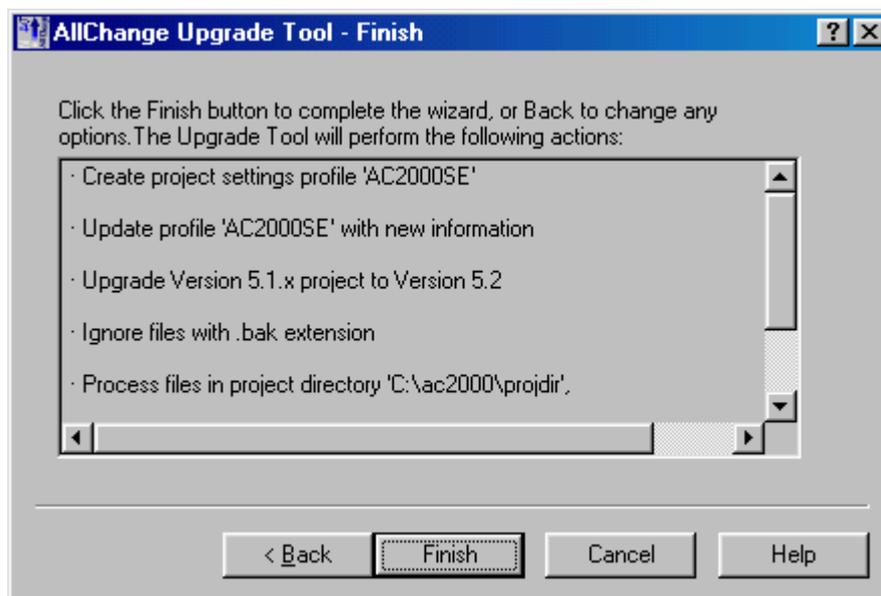
**Stage 3:** post process the database, see [Stage 3](#)

Note that all stages may take a significant amount of time and in particular stage 2 may require a significant amount of administrator interaction.

You may select to auto-process or you may manually do so later.

### New upgrade session — Final Screen

This is the final screen and informs you of what ACUPGRAD will do.



You may select **Back** to review or modify any selections on the previous screens or select finish to start the upgrade process and perform any automated processing selected on the **Options** screen.

On selecting **Finish** the session information will be saved in a file in the project directory with the same name as the project and a `.ugs` extension. Information is saved as to the project and system directories etc. as specified in the wizard as well as information as to how much of the upgrade process has been completed. Whenever a new item has been upgraded this information will be stored in the session file. See [Upgrade Sessions](#) for further information.

### Continuing an Upgrade Session

Having selected to continue an upgrade session you will be prompted to select the session information file (`.ugs` file) for the session you wish to continue, see [Opening a Session](#).

You may then continue with the upgrade process.

### Take a project live with a previous upgrade using AC4AC

If you are using **AllChange** to control the configuration files for your projects (AC4AC, See [AC4AC Project](#)), then having Gone Live with at least one project you may use this option to take another project (e.g Play or

Training) live using the same configuration files.

Having selected this option you will need to select your Old Project directory as for a new session, see [New Upgrade Session Old System](#).

You will then need to select the project to be upgraded. Projects in the old system directory will be listed.

On successful opening of a session for go live you will be presented with a file list to process as in the normal case. See [Going Live](#) for details of the go-live process.

## Upgrade Sessions

### *About Upgrade Sessions*

You may well find that you do not complete a whole project upgrade in a single session with ACUPGRAD.

This is not a problem as you can save session information as to how far you have got and then continue from there next time you start ACUPGRAD. Session information is automatically saved as you progress through the upgrade process, however you may use the **Save** and **Open** session options of the **File** menu to perform these operations as/ when you require.

### **Saving a Session**

Use the **File | Save Session** or **File | Save Session As** options to save information as to the current state of your current upgrade session.

The information is stored in a file with a `.ugs` extension, and you should choose to place it somewhere you will easily be able to find it again.

### **Opening a Session**

If you wish to continue upgrading a project that you started in a previous ACUPGRAD session then you should use the **File | Open Session** option.

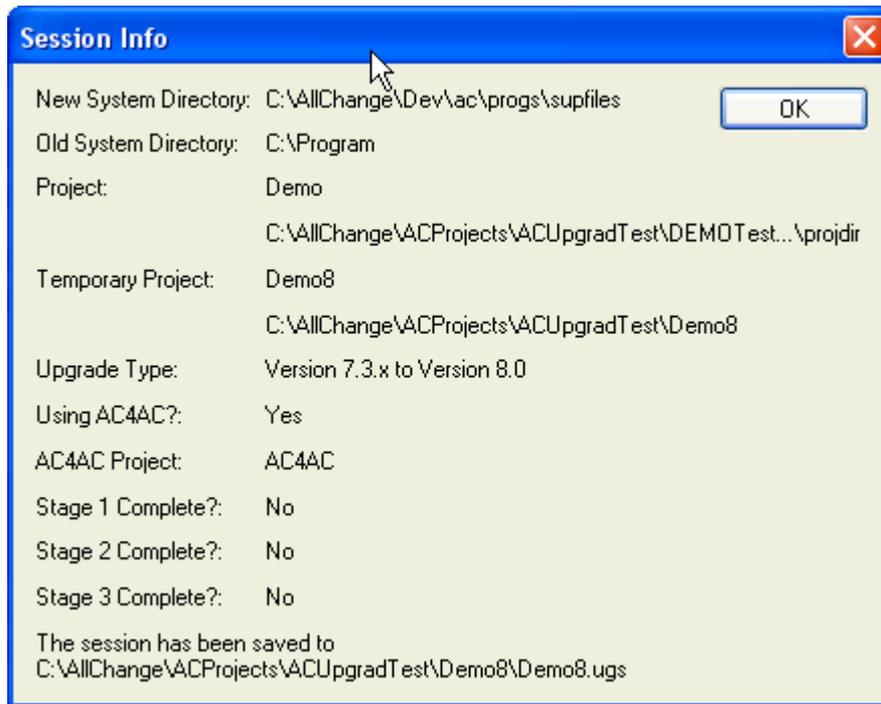
This will prompt you to specify the `.ugs` session file in which you saved the state of your upgrade session.

Information will be retrieved from the specified `.ugs` file about the intasoftini files and directories relevant to the upgrade as well as information as to how much of the upgrade process has been performed.

You may then continue from where you left off.

### **Session Information**

This shows you information about the current upgrade session.



## Performing the Upgrade Stages

### *About Performing the Upgrade Stages*

The three stages of the upgrade process are available from a button bar at the top of the main window. The current stage is shown highlighted, other stages are disabled/greyed out. From this button bar ACCONFIG and ACE may also be invoked. These are also all available from the **File** menu.

If any stage may not be performed it will be disabled. For example, stage 3 may not be performed until stages 1 and 2 have both been completed.

See also [The Upgrade Process](#).

### Stage 1

Selecting **File | Stage 1: Pre Processing** or the **Stage 1** button will perform the preprocessing requirements, this will usually involve a database conversion and may take some time depending on the size of your database.

This stage may be repeated if required or if an initial attempt failed.

Before attempting this stage you should ensure that the database to be converted is not currently in use i.e. you should have locked users out.

There may be some prerequisites of the database to be converted prior to performing the conversion. The [Check Database Integrity before conversion](#) option may be used to check for any prerequisites.

During the initial upgrade phase of creating a test project, stage 1 will copy the database to create a new Test database and convert it. During the Go Live phase of upgrading a project the live database is converted.

After the database has been preprocessed, during the initial phase of creating a test project, you will be prompted to copy some or all of the VC Files correspond to the database to the test project. You may wish to only copy a subset of these for testing purposes, but note that only those parts for which you have copied the VC Files will be able to be fully tested, e.g. with check in/out. If you wish to copy additional VC

**Copy VC Files.** During the Go Live phase of upgrading a project this is not necessary.

Under some circumstances it may not be necessary to perform a database conversion, for example, some minor upgrades. The Upgrade notes will inform you if this is not required. In this case selection of **Options | Advanced | Don't convert database** will cause the database conversion to be skipped when carrying out Stage 1 of the Upgrade.

## Stage 2

Stage 2 involves processing the configuration files which need to be upgraded. The files are shown in a list together with information as to the process action required and a tick box to indicate if the file has been processed.

If using AC4AC to control your projects the files are shown colour coded to indicate their location and AC4AC status:

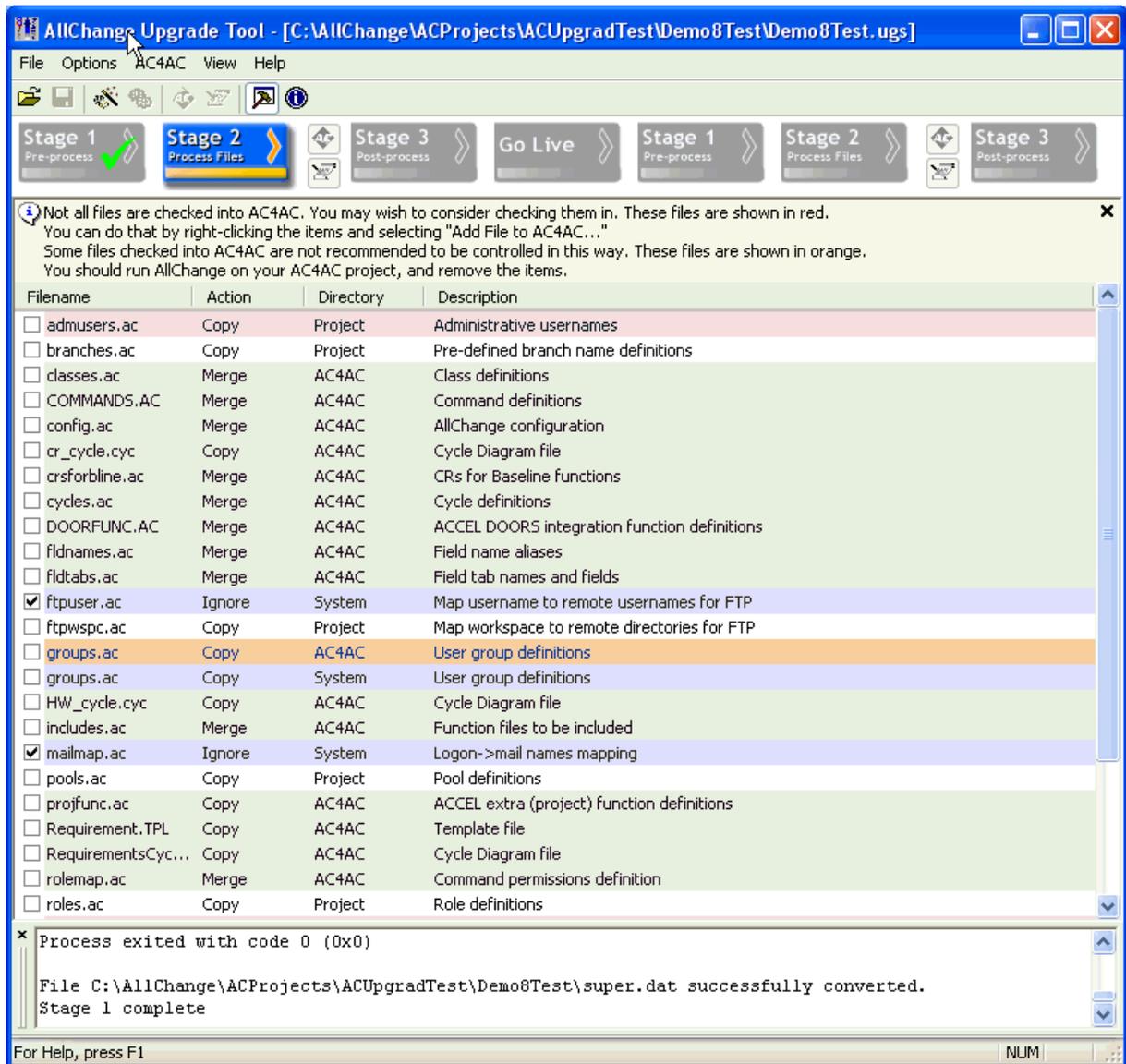
**White:** the file was found in the project directory, and is (correctly) not in AC4AC. If not using AC4AC, all files are displayed in white.

**Blue:** the file was found in the system directory

**Red:** should be kept in AC4AC but is not, or should not be in AC4AC but is (only if using AC4AC). An explanation message is displayed if any items are in this state. See [AC4AC Project](#) for files which are not recommended to be stored under AC4AC control. Items in red may be added to AC4AC using **AC4AC | Add File to AC4AC**.

**Green:** the file is in AC4AC.

**Grey:** the file has an action of ignore, and is not coloured to indicate one of the above states.



The conversion process for the configuration files may involve any of the following types of operation:

- Copy:** Simply copy the file to the new project directory i.e. it requires no upgrading/ conversion. If the file is under AC4AC control it will be checked out of AC4AC.
- Delete:** The file is no longer required and so will not be copied to the new project directory.
- Merge:** Any site specific changes need to be re-applied (*merged*) into the new supplied standard file. By default the **AllChange VisMerge** tool is used to perform this process, see **AllChange User Manual** for details of the merge tool. The merge tool to be used may be specified as a third party tool such as Araxis Merge using [Options | Merge Tool](#). If the file is under AC4AC control then it will be checked out from AC4AC. If **Perform silent merges** option is selected, then the visual merge tool will only be invoked if there are clashes,
- Convert:** The file requires some specialised conversion. Note that this may involve another operation such as a **Merge**. If the file is under AC4AC control then it will be checked out from AC4AC.
- Ignore:** This will be present for files in the system directory which have already been processed (e.g. when upgrading another project) and so do not need to be processed again, or for files which do not need processing when going live (e.g. workspcs.ac)
- Check-out:** This will be set for the Go Live process when configuration files are under AC4AC con-

trol.

The file list may be refreshed using **Options | Refresh File List** and the AC4AC status may be refreshed using **AC4AC | Refresh AC4AC Status**.

For the *go-live* process, you should ensure that any configuration files which are not project specific have been removed prior to processing files in stage 2, i.e. all files should be removed except log files, workspace and pool definitions, role and group definitions which are project specific, branch and part permissions. This ensures that any obsolete files are not left around after the upgrade process and that the *live* project directory contains only the same files as the *test* area where the configuration files were upgraded and any testing has been performed. *Note that the upgrade tool will offer to do this for you.*

If using AC4AC, ACUPGRAD will create a new workspace definition for the AC4AC project with a workspace directory of the test project directory. As each file which is controlled by AC4AC is processed:

- when upgrading to the temporary project, it is checked out for edit from AC4AC to the workspace for the temporary project, prior to any necessary processing (e.g. merge).
- When going live each file is checked out read only from AC4AC to the workspace for live project directory

Any files present in the old project directory which are not known to ACUPGRAD will be shown with a **Copy** action.

This list is shown automatically on completion of the wizard and whenever a session is opened.

Each file may be processed in turn by double clicking on the file. Alternatively, **File | Stage 2: Process Files** will process any selected files.

Selecting **Stage 2** on the button bar or **File | Stage 2: Process All Files** will automatically process each file in the list in turn — note that this may require user interaction.

The processing will perform whatever action is specified in the list.

If the action is **Merge** (or possibly **Convert**), the **AllChangeVisMerge** tool will be invoked to allow you to merge your site specific changes in the **Old Project** into the new standard supplied configuration file (from the **New System Dir**). The resulting merged file will be placed in the **New Project Directory**.

You may override the use of the **AllChange VisMerge** tool and specify your own tool to use for merging using **Options | Merge Tool**.

If there are any clashes which occurred as a result of the merge then these must be resolved. A clash occurs when there are site specific changes to the same area of the file as changes made between the old standard supplied file and the new standard supplied file. You will have to make a decision as to which changes you wish to keep, yours, Intasoft's or both. In many cases you may find that your site specific changes are made redundant by the changes made to the standard file. Wherever possible you should retain the standard supplied changes.

In **VisMerge** the **Left file** will be the **New System File**, the **Right file** will be the **Old Project file**. For full details of **VisMerge** see **AllChange** User Manual.

Having completed a merge, saved the result and exited **VisMerge** the file will be ticked as processed.

If you exit **VisMerge** without having resolved all the clashes, you may save the current merge state if you wish to continue with resolving them. When you return to ACUPGRAD you will be asked if you wish to mark the file as complete or not. If you mark it as complete, you should finish resolving the clashes before proceeding to Stage 3. If you do not mark it as complete the entire merge may be started again if required, alternatively you may continue to resolve the clashes and mark the file as processed when this is complete, before progressing to Stage 3.

In order to continue with a merge session you should ensure that you save the merge information (in a `.mrg` file) before exiting **VisMerge**. When you wish to continue the merge session simply invoke the **VisMerge** tool (e.g. from **ACEFile | Merge**) and open the `.mrg` file that you saved previously. You may then

continue to resolve the clashes. When all are resolved the resulting file should be saved in the new project directory and the `.mrg` file may be deleted.

After all the files have been processed in Stage 2, **AllChange** may be run on the new temporary project, if it fails then any errors must be resolved prior to progressing to Stage 3. Both **AllChange** and `ACCONFIG` may be invoked from `acupgrad` using the button bar or from the **File** menu.

### Stage 3

Stage 3 involves any post processing that may be required after both stages 1 and 2 have been completed. This usually involves further processing of the database to make it compatible with the newly converted configuration files.

This process may be lengthy depending on the size of the database to be converted.

Prior to performing stage 3 for a new temporary project you should ensure that you:

1. Copy the VC files required - this is prompted for after Stage 1 database conversion, or may be manually performed using **File | Copy VC Files**.
2. Copy the attachments directories if required
3. Modify any workspace , pool or ftpworkspace definition

This is *not* necessary for the *go-live* process.

See [The Upgrade Process](#).

Note that if your database uses absolute **Location** fields for subsystems or for your root (`/`) part then you must be sure to modify these manually before testing the new configuration.

You will then be prompted for which post processing options you wish to perform. These will vary depending on the versions you are upgrading from and to.

### Options

The **Options** menu provides various options and information as follows:

#### Refresh File List

Causes the list of files shown to be refreshed from the filing system.

The results of all processing are displayed in this window.

#### Merge Tool

Allows the Merge tool to be used to be specified. This should be specified as the path to the executable file for the tool. If unspecified the default supplied tool `VisMerge` will be used. If a third party tool is to be used it should be a three-way merge tool, and must be able to be run with a command-line which looks like:

```
<path to merge tool> <options> <file1> <file2> <file3>
```

Options for the tool may be specified after the tool's path. Options should appear outside any quoting of the path.

If using `Araxis Merge`, you may specify either `merge.exe` or `compare.exe` as the merge tool, though we would recommend `compare.exe`, as it is better suited to the purpose. If using `compare.exe` you do not need to specify the `'/wait'` or `'/3'` command-line arguments as `acupgrad` will add them.

`Araxis Merge` should be installed in a path which contains the word **Araxis** in order for the Upgrade tool to recognise it.

#### Perform silent merges

If selected, then when a configuration file is merged and no clashes are encountered, the merge tool is not invoked and the file is taken as processed. If a clash occurs then the merge tool will be invoked to allow the clashes to be resolved.

**Mark Stage 1 as Complete:** this will flag **Stage 1** of the upgrade process as complete as far as the upgrade tool is concerned. You may wish to do this if, for example, you have performed the pre-processing yourself outside of ACUPGRAD and manually copied the VC Files to the temporary project directory

**Check Database Integrity before converting:** this will cause the upgrade tool to check the database that is to be upgraded for any pre-requisites prior to upgrading/convert it. For example, when upgrading from version 8.1 to 9.0 it is a prerequisite that all database entries are case-insensitively unique, this option will check that this is the case prior to converting the database.

**Dont convert database:** this will flag **Stage 1** of the upgrade process that there is no need to convert the database. You may wish to do this if, for example, you have performed the preprocessing yourself outside of ACUPGRAD but you still wish other parts of Stage 1 to be performed such as copying the VC Files.

**Mark selected file(s) as complete:** this will flag configuration files as having been processed (i.e. they will be ticked in the file list). You may wish to do this if, for example you performed some processing but did not save the session information and wish to continue and not reprocess the files already processed.

**Mark selected file(s) as NOT complete:** this will flag configuration files as *not* having been processed (i.e. they will be un-ticked in the file list). You may wish to do this if, for example you performed some processing but did not complete the job and so which to reprocess some files already processed.

**Mark Stage 3 as Complete:** this will flag **Stage 3** of the upgrade process as complete as far as the upgrade tool is concerned. You may wish to do this if you wish not to perform any post processing on the database, but continue with the upgrade process.

**Don't convert database:**

This prevents stage 1 from performing any database conversion.

**Remove file(s) from list:** This removes the selected item(s) from the list of files to process. If a file is removed from the list it will not be included as part of the upgrade process. If a file is removed from the list it may be re-added later.

**Re-add removed files:** This adds back into the list of files to process any that have been removed from the list.

**Override Action:** this allows the action for processing a file to be changed.

**Ignore File:** causes file to not be processed. When processing all items, or checking to see if all items have been processed, items marked as "Ignore" are not considered. This is useful if the file is not required at the moment in the upgraded project, but may be needed later on in the upgrade process.

**Copy File:** causes action to be changed to a Copy, which simply makes the upgrade tool copy the file to the new project directory with no processing.

**Restore to default:** causes the action to be set back to the default action.

## Going Live

When all testing has been satisfactorily performed on the upgraded test project then you can *go live* with the new version of **AllChange**.

In order to do this you will need to follow the steps outlined at the start of this chapter.

The **File | Go Live** option or the **Go Live** button (or the go-live option in the wizard) will perform the following operations:

- A project definition for the new **AllChange** will be created based on the old project definition for client/server and other properties. These may be modified using ACCONFIG after the go live process if required.

- The definition of the project in the old system will be removed to prevent inadvertent attempts to access it using the old software (e.g. users who have not yet done a workstation install for the new system)
- It will prompt you to delete files from the live project directory which are to be replaced by the upgraded files or which are not known/wanted for the upgraded live project. If using AC4AC, then any files controlled by AC4AC should be checked out read only to the workspace for the live project directory, these are checked in. Any files which are no longer exist in the temporary project are also made obsolete in AC4AC.
- If using AC4AC to control your project configurations it will prompt you for a baseline for the upgraded configuration files which are to be checked in from the temporary upgraded project. The configuration files will then be checked in and added to the baseline specified if any. The files will then be left checked out read only to the workspace for the temporary project so that this project is still able to be used.

Before going live you should ensure that no-one is using the live project. You can use [Lockouts](#) to prevent any users from using the system.

All 3 upgrade stages need to be completed in order to go live.

- Stage 1:** this will process the *live* database in order to convert it to the new format
- Stage 2:** this will allow you to copy all the configuration files which have been upgraded in the test/play area into the original project directory. Files which are controlled by AC4AC will be checked out read only from AC4AC into the workspace for the live project directory. Files which are not controlled by AC4AC or if not using AC4AC will be copied from the upgraded test project. It will also allow you to process the configuration files which require re-processing in place, if any, such as **workspaces, pools, ftp workspaces**. *Before performing stage 2 you should remove all configuration files which are to be copied from the test project and any files which are no longer required from the live project directory. The upgrade tool will offer to do this for you on selecting to Go Live.*
- Stage 3:** this will perform any necessary post processing on the *live* database. This stage may need information from some of the original unconverted configuration files, these will be taken from the `upgrade.bak` directory. **Before performing stage 3 you will need to ensure any Lockout permits Administrator access in order to allow the upgrade tool to access ACC to perform the post processing. Afterwards you may wish to continue to deny non Administrator users access until you have performed any checks you wish to perform before allowing users access again.**

Users will probably need to perform a workstation install to be able to use the new upgraded project.

After **Stage 3** is complete ACUPGRAD will give you the opportunity to clean up the temporary project, this will remove the project definition delete all the content of the project directory and delete the database. If using AC4AC the files will be checked in from the workspace for the temporary project directory.

If you do not choose to clean up the temporary project at this time, this may be done using **File | Clean Up Upgrade Directory**.

If using AC4AC you will be offered to take another project live with the same configuration files. This may also be invoked from **File | Go Live With Other Project** or from the Wizard on startup of ACUPGRAD. This will perform the Go Live process for an additional project (e.g. Play or Training) using the configuration files in AC4AC and processing the database for the project, i.e. you must do all 3 stages of the Go Live process for the other project.

## Creating a Copy of an AllChange Project

A copy of an **AllChange** project may need to be made for a number of different reasons, for example:

1. In order to upgrade to a new release of **AllChange** without disrupting users whilst the upgrade takes place.
2. In order to perform additional configuration or reconfiguration of an existing project without disrupting users whilst the reconfiguration takes place.
3. In order to create a training or *play* area which mimics the live project.

The copy may be created either *manually* or may be partially automated using the **Upgrade tool**.

In either case the following steps need to be performed:

- Ensure no users are using the **AllChange** live database whilst it is being copied (e.g. use the **Lock-out** facility).
- Create a new project definition and copy the project configuration files into the new project directory and create a copy of the database for the new project .

*ACCONFIG will do this for you if you create a new project definition by copying an existing one.*

*The upgrade tool will do this automatically for you performing any conversions necessary.*

- Copy the original projects VC files to the new project directory e.g.

```
XCOPY /S/E \ACPROJDIR\VCFILES \ACPLAYGD\VCFILES
```

*The upgrade tool will do this automatically for you*

- Users may be allowed to use the live system again now (modify/remove the lockout).
- Invoke ACE and select the new project you have just created (or create a shortcut if not using project definitions).
- Alter the location of any parts in the new database with absolute paths to point to the new VC file location. This only needs to be done if using non default absolute paths for the location field of any parts. If the default location of the root part only is used then non change is required.

```
alter location=X:\VCFILES /
```

*The upgrade tool will alter the root location for you, but if you have any absolute locations on other parts then you should modify these as appropriate.*

- Alter the workspaces and pools in the new project to reflect different locations than the workspaces for the project copied.

# ACCEL --- AllChange Command Evaluation Language

## About ACCEL — AllChange Command Evaluation Language

**AllChange** includes a *command evaluation language*, 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.

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 [Command Definitions in Depth](#)
- when writing user-defined functions
- when specifying filters on what items to show in ACEwindows
- when tailoring the ACE Front-end to site-specific requirements by "greying out" or setting the behaviour of items under certain circumstances as described in [Menu Item/ Toolbar Availability](#), and [Intercepting Viewer Buttons](#). It is also used for configuring the ACCEL condition editor as described in [ACCEL Condition Editor](#).

ACCEL provides:

- access to all fields of all databases
- hooks to **AllChange** functions and commands
- evaluatable conditions
- string, list and arithmetic manipulation
- variables
- user defined functions
- operating system calls, file handling, communication with other programs
- functions to interact with the user from ACE (see also Visual ACCEL — [Visual ACCEL](#))

Where ACCEL conditions may be entered inside ACE the ACCEL Condition Editor is available to aid in the task. This is a dialog which allows ACCEL items to be selected from drop-down lists and restricts choices to those pertinent to the context.

### Uses of ACCEL

**baseline**, [command definitions](#), **find**, **report**, **eval**, [life-cycles](#).

### Notation

When describing ACCEL the syntax will be expressed in the following notation:

A rule will have the form:

$$\begin{array}{l} lhs ::= rhs \\ \quad | rhs \\ \quad \cdot \\ \quad \cdot \\ \quad \cdot \end{array}$$

The *lhs* has the form:

<name>

The  `::=`  indicates that various alternative definitions for the rule now follow. Each alternative is separated by the  `|`  character.

Each *rhs* will have the form of a sequence of *rhs* elements which may be any of the following:

- literal text:** this is a sequence of ASCII characters (excluding a space) and is actual text that forms part of that construct
- non-terminal:** (i.e. `<name>`) this refers to another rule
- [elements]:** this indicates that the items inside the `[ . . ]`'s are optional
- {elements}:** this indicates that the items inside the `{ . . }`'s may be repeated as many times as required.
- # any text:** this is a comment

## ACCEL Expressions

ACCEL allows the specification of *expressions*. Formally an expression is:

```
<expression> ::= <statement> { ; <statement> } [ ; ]
```

An expression is one or more statements separated by a  `;`  (semi-colon) and optionally terminated by a  `; .`  It does no harm to have a semi-colon when one is optional, so it may be easiest to terminate all statements with a semi-colon. The result of an expression is the result of the last statement evaluated.

## ACCEL Statements

A statement in ACCEL is an:

- [each](#) statement
- [foreach](#) statement
- [while](#) statement
- [if](#) statement
- [traperror](#) statement
- [return](#) statement
- [a condition](#).

Formally a statement is:

```
<statement> ::= <each-statement>
| <foreach-statement>
| <while-statement>
| <if-statement>
| <traperror-statement>
| <return-statement>
| <condition>
```

## Each Statement

The `each` statement is used to perform loops over items in a database. Formally this is:

```
<each-statement> ::= each <control-list> do
                    <expression>
                    endeach
```

where `<control-list>` is a list (or space-separated string) of the form:  
`<database> [<flags>] [<items>]`

### Database

`<database>` is compulsory and identifies which database to iterate over; this is a "user-friendly" database name, as accepted by the `report` command or the [num\\_records\\_in\\_range\(\)](#) ACCEL function. It may be one of:

Main Databases	Items Affected Databases	Status Log Databases	Configuration Data	No Database
part version issue instance cr baseline baseline_ detail part_vote cr_vote baseline_ vote monitor	part_items_ affected cr_items_ affected baseline_items_ affected	part_status_ log cr_status_log baseline_ status_log	branch class command config cycle cycle_ status field_name field_tab group include partperm pool report_for- mat role username vote_def- inition workspace	none

Note that the check-outs log database is known internally as the issues database.

Flags

<flags> may be used to modify behaviour of the loop through the database. The flags that may be used and the databases to which they apply is shown below:

Flags	Databases	Description
-all -reg -edit -def -top	part; ver- sion; base- line; baseline_ detail; cycles	Determines which versions of parts are required
-uses	part	If specified then all usage relationships will be followed to the part used
-recursive	part; ver- sion; issue; baseline; part_status_ _log	Causes all members of a part/baseline tree to be included in the loop
-recurseup	baseline_ detail	May only be used in combination with -indexedby part-name
-latest <number>	baseline	Limits the number of versions of each baseline
-progress	all	display a progress bar
-indexedby <index>	all	specifies the name of the database field(s) to be used to filter/sort by. The field name must be the 'real' database field name as returned by the <a href="#">index_names</a> function. Multiple fields may be specified separated by / character. e.g. assignee/number will cause a sort/filter by assignee first and then cr number.

-allowabort or -allowescape	all	allows the user to abort the loop using an abort dialog or Esc key
-crbuffer	cr	allows cr browser buffered reading of the cr database
-direction <direction>	cr	determines the direction of the buffered reading of the database (next/previous) buffer - see <a href="#">Reading the CR Database Buffered</a>
-itemaffectedtype <type>	part_items_affected; cr_items_affected; baseline_items_affected	specifies the type of the item affected when used in conjunction with -indexedby itemaffected. <type> should be cr, part or baseline.
-solved	cr_items_affected	if the versions solved relationship is required the this should be specified
-field <name><op><value>	all	argument to filter database records at the server. This argument is detailed <a href="#">below</a> .
-nolinkedrecords	all	specifies that no linked (related) records should be fetched. this argument is detailed <a href="#">below</a> .

Please see also [Databases](#)

### Items

<items> may be used to pick out individual item(s), or a range of items; if no <items> are specified the behaviour is the same as with **report** (current working part downward on the parts database, all entries on other databases). Items may use \* to match wildcards e.g. starts-with\*, \*ends-with, \*contains1\*contains2\*

The each loop functions similarly to control levels in report formats and a detailed description of databases, flags and items can be found in [Report Format Syntax](#).

For each item in turn, ACCEL sets the current record (see [ACCEL Field References](#) below) to that item (allowing plain field references like pa\_part, cr\_number etc.) and then executes <expression>. The primary use is for script writers to be able to look through many elements in the databases; it is also possible to utilise this mechanism on a single record when there is no current record simply to avoid qualified references which are slow.

The result of an each statement is the last expression evaluated.

### Specifying Fields to an 'each' statement

The each statement accepts an argument in the form:

```
-field <name><op><value>
```

to allow records to be filtered by the server where the specific *items* required are not specified or contain some kind of wild card. This can significantly improve performance, as it minimises the number of records passed from the server to the client over the network.

Note that where specific *item*(s) are enumerated the -field argument is ignored. For example, if using the each statement for a single CR or a specified list of CRs, the -field argument should not be used, if statements should be used instead to make any qualification needed.

Multiple field arguments will be AND'ed.

The parts of this argument are specified as follows:

<name>:

a fieldname, as specified in the SQL database (SSMS can be used to view these). Applies to

the "primary" database being visited; do not use with **-recursive** or **-all**. Note that CR, part and baseline text may be searched using `text` as the field name.

**<op>:**

an operator to use when determining field matches. This may be one of:

`==, !=, >, >=, <, <=` with the usual ACCEL semantics

`=, <>` as SQL-style synonyms for `==` and `!=`

`%` for LIKE or contains

`!%` for NOT LIKE

`@` for IN or one-of (value is then a single-space-separated list)

`!@` for NOT IN

**<value>:**

the string to be sought. This value is always case-insensitive. A left-hand-side match will be used if the value ends with `*`. For 'truth' fields the value should be 0 or 1. The value can be quoted if necessary (especially for `@` operator).

**Examples:**

suppose you have a loop which wants to visit every check-out-for-edit in a workspace which has CR00001 in its comment. Without using the field argument, you may use:

```
each 'issue -indexedby wspcname '.workspace do
  if iu_foredit and match_wild(iu_comment, '*CR00001*') then
    ...
  endif;
endeach;
```

This can be made more efficient -- if there a large number of check-outs to the workspace which do not satisfy the 'if' -- by using:

```
each 'issue -field foredit=1 -field comment%CR00001 -indexedby
wspcname '.workspace do
  ...
endeach;
```

If an **each** statement is used to simply make a single record the "current record" so that within the body of the **each** statement fields of a record can be accessed as "plain" references rather than having to repeat the qualification e.g.:

```
each 'CR '.var(Global_cr_just_created) do
  if cr_class == 'SCR' then
    ...
  endif;
endeach;
```

In these cases any **-field** argument is simply ignored, since no "looping/searching" is done at the server, so its use here would not be appropriate.

See the supplied ACCEL function files for more examples of the usage.

### *Linked Records*

Normally when executing an `each` loop to fetch records from a database the `each` additionally pre-fetches corresponding records from *related* databases, in anticipation that they will be needed. For example, an `each` on the issues database also fetches the corresponding versions and components. This can result in a *large* time saving (especially on a slow connection). If you know that you will **not** be accessing any related records you can disable this behaviour (and achieve a modest time-saving) by specifying `-nolinkedrecords`. For example, the following "each" does not require any related part/version records:

```
each 'issue -nolinkedrecords' do
  echo(iu_date);
endeach
```

### Reading the CR Database Buffered

When iterating over the CR database **-crbuffer** and **-direction** *<direction>* may be used to implement buffered reading of the CR database.

These permit the **each** loop to behave similarly to the way the CR Browser does, viz. reading only up to a buffer's worth of CR records at a time and allowing next/previous buffers to be read from where last reached.

Specifying **-crbuffer** causes the CR buffer reading code to be executed.

*<direction>* should be a number: 0 (zero) first time to read the initial buffer's worth (which is actually the last buffer), subsequently -1 or 1 to go backward or forward respectively by a buffer's worth. The actual determination of when a buffer is filled must be made by the ACCEL code within the **each** loop counting and executing a **return** statement to exit the function it is in when the buffer is full, e.g. a buffer's worth of open CRs could be read by:

```
setvar(count, 0);
each 'cr -crbuffer -direction '.var(direction) do
  if status_is_open(cr_class, cr_status) then
    echo(cr_number);
    setvar(count, var(count) + 1);
    if var(count) >= var(CRBufferSize) then
      return;
    endif;
  endif;
endeach;
```

These flags are *only* intended for implementing a CR Browser-type buffered read from an **each** loop. They are used to supply CR Browser functionality from a Web browser. Only one **each -crbuffer** loop may be active at a time (i.e. they do not nest).

### Foreach Statement

The **foreach** statement is used to iterate over a number of items evaluating an expression on each item in turn. Formally this is:

```
<foreach-statement> ::= foreach <var-name> in <condition> do
    <expression>
endfor
```

*<var-name>* should be a user-defined variable name. *<condition>* is evaluated and converted to a list; the variable is then set — just as in the [setvar](#) function — to each element in the list in turn and *<expression>* is evaluated. The variable's current value may be accessed within the *<expression>* using the [var](#) function.

The result of a **foreach** statement is the result of the last expression evaluated.

### While Statement

The **while** statement is used to perform loops. Formally this is:

```
<while-statement> ::= while <condition> do
    <expression>
endwh
```

While *<condition>* evaluates as true *<expression>* is executed.

The result of a **while** statement is the result of the last expression evaluated.

### If Statement

The **if** statement is used to express decisions. Formally it is:

```
<if-statement> ::= if <condition> then
```

```

    <expression>
  {elsif <condition> then
    <expression>}
  [else
    <expression>]
  endif

```

It may contain any number (including zero) of `elsif` [*sic*] parts; the `else` part is optional, but must come after any `elsifs`. The `<condition>` is evaluated and if it is true then the following `<expression>` is executed. If it is false and there are any `elsif` part then each of these `<condition>`s is evaluated in turn until one is true, whereupon the following `<expression>` is executed. If all `<condition>`s evaluate as false and there is an `else` part then its `<expression>` is executed.

The result of an `if` statement is the result of the last expression evaluated.

## Traperror Statement

The `traperror` statement is used to allow errors to be caught and acted upon in ACCEL code. Formally it is:

```

<traperror-statement> ::= traperror
    <expression>
    onerror
    <expression>
    enderror

```

The statements between `traperror` and `onerror` are executed normally; if no error occurs execution jumps immediately to the first statement after `enderror` as soon as it reaches `onerror`: the `onerror` statements are ignored.

If an error occurs between `traperror` and `onerror` execution jumps immediately to the first statement in the `onerror` block. Instead of the error being shown to the user, the text of the error message is placed in the ACCEL variable `error_message` (see [ACCEL Variables](#)). The statements up to the `enderror` are executed; if no error is raised execution then continues at the statement after the `enderror`. Typically code in the `onerror` block is used to clear up after an error in the `traperror` block; the error may then be reraised by placing an `error(error_message)` statement just prior to the `enderror`, e.g.:

```

setvar(fin, '');
traperror
  setvar(fin, fopen('infile', 'r'));
  setvar(line, freadline(var(fin)));
  fclose(var(fin));
onerror
  if var(fin) != '' then # got past the fopen()
    fclose(var(fin));
  endif;
  error(error_message); # reraise the error
enderror;
echo("File read successfully");

```

Alternatively the error may be completely suppressed and execution allowed to continue — but be careful that the error is safe to ignore in this case — by *not* putting an `error()` statement in the `onerror` block. In this case, do not forget that since the error message has not been shown to the user it would be advisable to display it (unless you really mean not to), e.g.:

```

traperror
  copyfile('from', 'to');
onerror
  message_box(error_message, 'Error during copyfile()', 0);
enderror;
echo("File may or may not have been copied - don't care");

```

Error trapping may be nested, including across any user-defined functions (see [User-defined Functions](#)) called from a `traperror` block. Errors are always raised to the latest enclosing `onerror` block, if any, regardless of whether the code called has no error handling of its own or has handled and then reraised an error, e.g.:

```
Function_1
action
  traperror
  call(Function_2);
  onerror
    echo('Function_2 raised an error');
    error(error_message);
  enderror;
  echo('Function_2 did not raise an error');
end
Function_2
action
  echo('Might have an error here...');
  traperror
    echo('...Or might have an error here...');
  onerror
    echo('...in which case we can clear our stuff up and then...');
    error(error_message);
  enderror;
end
```

Any errors raised while in an `onerror` block are raised to the next enclosing `onerror` block, if any (unless the `onerror` block has its own `traperror` statement).

## Return Statement

The `return` statement is used to return prematurely from the block (condition or action) in which it appears. Formally it is:

```
return [<condition>]
```

It may be used inside a loop to terminate the iteration: this makes it possible to write functions which, say, stop at the first status log for an item without having to read all the other status log records. The `return` may be followed by a condition, in which case that is the value returned, otherwise no value is returned. `return` makes commands easier to write and read.

## ACCEL Conditions

Conditions form the core of ACCEL. They are relational expressions and always have a value as their result.

The result of a condition may be of several different types:

- true/ false
- string: a sequence of ASCII characters
- list of strings
- numeric

A condition is made up of a sequence of `<factors>` and `<operators>`, where the `<operators>` are applied to the `<factors>`.

The `<factor>` may be any of the following:

- field-reference:** this is a reference to a field in one of the *AllChange* databases; see [ACCEL Field References](#).

<b>built in function:</b>	this is a call to a built in function, see <a href="#">ACCEL Functions</a> .
<b>built in variable:</b>	this is a reference to the value of a built in ACCEL variable, see <a href="#">ACCEL Variables</a> .
<b>literal-string:</b>	<p>this may be any sequence of ASCII characters. If it includes a space or a punctuation character — or if it happens to be the same as another word used by ACCEL — then the string <i>must</i> be enclosed in ' (single quote) or " (double quote) characters; otherwise a simple word may appear without quotes, though it may be preferable to quote all literal strings anyway. Inside double-quoted strings (only) the ^ (circumflex) character may be used to cause certain special characters to be embedded in the string as follows:</p> <ul style="list-style-type: none"> <li>^n newline character</li> <li>^r carriage return character</li> <li>^t tab character</li> <li>^" double quote character</li> <li>^^ single circumflex character</li> <li>^ <b>3-digits</b> character represented by 3 (decimal) <i>digits</i></li> </ul> <p>(This feature is intended for use in message strings, e.g. for producing multi-line mail message bodies.)</p>

Everything in ACCEL ultimately resolves to a value of one of the different data types, where:

- a *truth value* is either *true* or *false*
- a *string* is a sequence of characters
- a *list* is a list of strings
- a numeric is a number

ACCEL automatically converts values to whatever type is appropriate to the context, so, for example, a function returning a string may be used in the condition part of an `if` statement (where a truth is required) and ACCEL will convert the string to a truth value.

The rules ACCEL uses when converting types are as follows:

- When converting from a truth to a string value, if the value is *true* then the corresponding string value is `true`; if the value is *false* then the corresponding string value is the empty string (' '), i.e. contains no characters).
- When converting from a truth to a list value, the corresponding values are a list with a single string of `true` in it and the empty list (no elements in it) respectively.
- When converting from a string to a truth value, an empty string evaluates to *false* while any other string evaluates to *true*.
- When converting from a string to a list value, the string is basically split on space characters to produce a list of strings, but subsections inside " (double quote) characters produce single elements (see the `splitquote()` function).
- When converting from a list to a truth value, an empty list evaluates to *false* while any other list evaluates to *true*.
- When converting from a list to a string value, the strings in the list are concatenated (joined) with a single space character in between each element to produce a single string (see the `join()` function).

ACCEL treats numbers and strings interchangeably, converting freely between them as expected, i.e. the string '10' and the number 10 are equivalent. Note that a floating point literal number such as 1.5 must be

quoted (e.g. '1.5') otherwise ACCEL will treat it as a string concatenation, as do negative numbers (e.g. '-1').

Any numeric strings that contain decimal points (or other character for decimal separator, as per user's settings) will be treated as floating point numbers, and any arithmetic carried out on them will be returned as floating point numbers. The results will be formatted to use the user's local decimal point character to 2 decimal places.

The ACCEL functions [format number](#) and [unformat number](#) can be used to format a number in the user's local settings, and take a number in the user's local settings respectively.

Certain operators work with numbers, such operators either require numerical values (e.g. addition) — in which case ACCEL always converts to numbers before performing the operation — or may accept either numeric or string values (e.g. comparisons) — in which case they perform a numeric operation if both values look like numbers or a string operation otherwise.

The `<operator>` may be any one of the following:

<code>and</code>	performs a logical and
<code>or</code>	performs a logical or
<code>not</code>	performs a logical not (unary operator)
<code>==</code>	performs a comparison for equality
<code>==!</code>	performs a <i>case-sensitive</i> comparison for equality
<code>!=</code>	performs a comparison for inequality
<code>!=!</code>	performs a <i>case-sensitive</i> comparison for inequality
<code>&gt;</code>	performs a comparison for greater than
<code>&gt;=</code>	performs a comparison for greater than or equal to
<code>&lt;</code>	performs a comparison for less than
<code>&lt;=</code>	performs a comparison for less than or equal to
<code>.</code>	(dot) performs a string concatenation operation
<code>+</code>	performs arithmetic addition
<code>-</code>	performs arithmetic subtraction
<code>*</code>	performs arithmetic multiplication
<code>/</code>	performs arithmetic division

All comparisons on strings are *case-insensitive* (so `'ABCD' == 'aBcD'`), except for `==!` and `!=!` which are *case-sensitive* (so `'ABCD' !=! 'aBcD'`).

A condition is evaluated from left to right with following precedence assigned to the operators (highest to lowest; parentheses — `()` — may be used to alter the order of evaluation):

Operator Precedence
<code>( )</code>
<code>* /</code>
<code>. + -</code>
<code>== ==! != !=! &gt; &gt;= &lt; &lt;=</code>
<code>not</code>
<code>and</code>
<code>or</code>

## ACCEL Field References

### About Database Fields

Each of the parts, CRs, check-outs, baselines, instances and monitors are regarded as a database. In addition there is a status log database which holds all the status log entries for parts, baselines and CRs.

*Note that the check-outs database is known internally as the issues database.*

Each database has a number of data fields associated with it. These will be of the following types:

string	plain text. These may be of varying size.
<a href="#">date/time</a>	date/time stored as text string in UTC and translated on input/output from/to local time as appropriate
truth	holds text string values true/false
<a href="#">numeric</a>	numbers stored in Intasoft format and translated on input/output to/from local format as appropriate
user	holds user names, these are stored as the user logon id and translated in input/output using the Full Name from the user registration

### *Date/Time Fields*

In order to support correct and consistent operation when data is accessed from multiple time zones **AllChange** stores date/times internally in UTC (GMT). (This is the only way processes in different time zones can agree on what a time *means*.)

This is largely transparent to users: although physically a date/time field in a record (e.g. the date/time a version was created) is stored in UTC, virtually anywhere the field is output (e.g. viewer, browser, report, ACCELfield reference) it is converted to local time, and virtually anywhere the field is input (e.g. viewer, command-line script) it is converted from local time. This behaviour applies to all built in date/time fields **and** to any arbitrary fields defines as of type date/time.

Unless stated otherwise, all **AllChange** date/times (including ACCEL functions) can be assumed to operate in local time.

Some points to bear in mind are:

- If, say, SSMS, is used to examine/patch database records they will be output/input in UTC rather than the local time used in ace .
- An ACCEL field reference to a field holding a date/time (e.g. pa\_date) does not deliver the same output independent of the calling process' time zone. For example, an **AllChange** user in the UK and another in the US who are looking at the same record in a viewer will not see the same time in the field — it will be adjusted to their local time zones. This is standard behaviour, e.g. the same is true in Windows Explorer. Most reports have been left outputting in local time. However, this is no use for archive/restore purposes; consequently our archiving reports and functions export/import the data in UTC. There are ACCEL functions to convert between UTC and local time if required.
- Log files (e.g. `acacts.log`) store time stamps in UTC.
- VC files store time stamps in UTC.
- The expansion in VC files of keywords containing times (e.g. `$ACheader$`) produces UTC.
- Lockout times in `lockout.ac` are stored in UTC. (ACCONFIG displays, and allows the administrator to enter, these times in its local time zone; be aware of storage in UTC if you edit the file directly.) This is required for cross-time zone access. This may cause a minor problem, not across time zones but between winter and summer time.

Suppose you have nightly backups scheduled for 3am. You might create a lockout to ensure no users are on-line between, say, 02:45 and 03:45.

This will be stored in `lockout.ac` in UTC (converted automatically by ACCONFIG). Unfortunately, your backup time of 3am may really be in local time (i.e. it's at 3am local time winter and summer): this means that in UTC terms it actually takes place one hour later in summer than winter.

If you leave it like this the lockout will come one hour earlier/later than the backup when daylight time starts/finishes (depending on whether you entered the time in winter or summer).

The "correct" solution is, when daylight time changes, to go into `acconfig` and change the time. The 02:45/03:45 you entered will now be displayed as either 01:45/02:45 or 03:45/04:45 (since it is

displayed in local time); change it back to 02:45/03:45 and this will be saved in the correct UTC. We realise this can be tedious, and suggest instead the time you originally enter for a lockout time should span an extra hour to cover for daylight time.

If you cannot work out which way, add an extra hour to both the start and the finish: in this case, create the lockout from 01:45 to 04:45 in the first place. We think this should be acceptable at most sites: leaving additional time overnight for the backup to execute is a good idea anyway.

- If a date/time is to be input from the user but only the date part is supplied, **AllChange** defaults the time to 12:00:00.

## Numeric Fields

In order to support correct input and output of real number for different local formats, **AllChange** stores real numbers internally in the Intasoft format of <num>.<num>.

This is largely transparent to the user: although physically a real number field in a record is stored in Intasoft format, virtually anywhere that the field is output (e.g. viewer, browser, report, ACCEL field reference) it is converted to local format (e.g. 1,23 in some European countries) and virtually anywhere the field is input (e.g. viewer) it is converted from local format.

### About ACCEL Field References

All of the fields of all of the databases may be accessed in ACCEL.

Formally a field reference has the general form:

```
<field-ref> ::= <field-ref-name> [( <factor> [, <factor>] )]
```

Each field reference name has the form:

*database-id\_field-name*

The *database-id* is a two character identification of which database and the *field-name* is the name of the field in that database which is required. The *database-id* for the primary data are `cr`, `pa`, `bl`, `iu` and `mo` for CRs, Parts, Baselines, Issues/Check-outs and Monitors respectively. An example field reference would be `cr_summary` for the CR database summary field

Depending on when an ACCEL field reference is made there may be one or more *current records*. In general only the field reference name need be specified when accessing a field of a current record; this is referred to as a *plain* field reference. Which records are current depends on the command/ action being performed:

- When reporting on a database each record in turn becomes the current record. Further records may become current in the course of the report. See the section on **report** in the **AllChange** User Manual. Also, the ACCEL `each` statement works in exactly the same way.
- When specifying a condition to certain commands (e.g. **report**, **find**, **baseline**) a certain record may be current. See under the particular command.
- When using the **eval** command no record is current. See the section on **eval** in the **AllChange** User Manual.
- When setting up conditions and actions for executing commands and progressing through life-cycles various records are current as appropriate to the context. See [Command Definitions in Depth](#).

Any field of any record in any (other) database may be referenced by following the desired field reference name by one or two parameters, in parentheses, giving the record to be sought; this is referred to as a *qualified* field reference. The database is searched for the first record whose key field(s) matches the parameter(s) supplied: this should refer to an item appropriate to the database as described under the **report** command, e.g. a part/version for the parts database, a CR number for the CRs database, a workspace name for the workspaces database, etc. Some fields of certain databases require two parameters in order to uniquely identify an individual record. Requirements for qualified references are noted in the field

reference tables below. Thus, when a record in the parts database is current `pa_type` gives the type of that record; when no record in the parts database is current `pa_type('/subsystem/part')` gives the type of the specified part, and similarly for the other databases. Note that a plain field reference is *far* more efficient than a qualified one since the former does not involve any searching of the database.

If a field is not *readable* by the current user, then a field reference will return the empty string in most cases. A field is unreadable if this has been specified in the appropriate [Command Access](#) *read* entries or an error is raised in the appropriate [Command Definitions](#) *read* entry. The exceptions to this are:

- when evaluating conditions in browsers and reports, this is to ensure that the condition is evaluated correctly
- when evaluating `accel` code in a [secure context](#)

Referring to a field of a non-existent record returns the empty string, list or *false*, as appropriate. Testing a field which cannot be empty (e.g. the key field) in a qualified reference is a suitable means of determining whether a record exists.

A few fields of certain databases are designated for holding "arbitrary" information. (Their field reference names are of the form: *database-id\_arbdigit*.) Such fields are intended to contain site-specific information. [Field Name Definitions](#) discusses how these fields may be assigned a more meaningful name. From ACCEL these fields may either be accessed by their fixed name, or by the field reference name assigned to them in the field name mapping file, e.g. `pa_arb1` or `pa_myfield`.

The following tables show each field reference for each database giving the type of value returned by the field and a short description.

### Field References for Parts Database

Information about each part known to **AllChange** is stored in the *parts database*.

#### Part Fields

Parts have a number of fields associated with them which may be used for querying/ reporting.

The fields are:

Title	Field Reference	Type	Description
<b>name</b>	<code>pa_name</code>	string	<p>The name of the part. Maximum 79 characters. The set of characters permitted in part names 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 <b>AllChange's</b> control. The characters allowed are:</p> <ul style="list-style-type: none"> <li>• Only ASCII printable characters (i.e. top bit not set, no control characters)</li> <li>• None of the characters <code>\ : * ? " &lt; &gt;   ^ = / ;</code></li> <li>• Must not contain <code>!!</code></li> <li>• The <b>first</b> character must not be one of <code>-@</code></li> <li>• Must not start with <code>\$\$</code></li> </ul> <p>Although the <code>'</code> (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.</p>
<b>parent</b>	<code>pa_parent</code>	string	The parent of the part. Maximum 239 characters
<b>part path</b>	<code>pa_part</code>	string	The full name of the part (i.e. parent and name fields, plus version

			if version)
<b>date</b>	pa_date	date/time	The date on which the part was created.
<b>obsolete</b>	pa_obsolete	truth	True/ false indication of whether the part is obsolete. true if the part is obsolete, false otherwise
<b>comp obsolete</b>	pa_comp_obsolete	truth	true if the component is obsolete, false otherwise (use when current record is a version)
<b>type</b>	pa_type	string	The type of the part ( subsystem , component or uses).
<b>class</b>	pa_class	string	The class of the part
<b>flags</b>	pa_flags	list	The flags set on the par. (Special characteristics of this part)
<b>no_file</b>	pa_no_file	truth	true if the no_file flag is set on the part, false otherwise
<b>no_vc</b>	pa_no_vc	truth	true if the no_vc flag is set on the part, false otherwise
<b>no_ver</b>	pa_no_ver	truth	true if the no_ver flag is set on the version, false otherwise
<b>dup_ver</b>	pa_dup_ver	truth	true if the dup_ver flag is set on the version, false otherwise
<b>locked</b>	pa_locked	truth	Part is locked against change. true if the part is locked, false otherwise
<b>comp locked</b>	pa_comp_locked	truth	true if the component is locked, false otherwise (use when current record is a version)
<b>status</b>	pa_status	string	The current status of the part.
<b>comp status</b>	pa_comp_status	string	The status of the component (use when current record is a version)
<b>location</b>	pa_location	string	The location of the part. The physical location of the operating system VC file/ directory corresponding to this part. Maximum 319 characters
<b>location raw</b>	pa_location_raw	string	The actual contents of the part's location field
<b>partsaffected</b>	pa_part-affected	list	The (other) parts affected by the part
<b>arb1--arb40</b>	pa_arb1--40	string, date/time, numeric	Arbitrary, site-specific information about the part. Arb1-3 and arb36-40 are limited to 120 characters; arb4-35 are limited to 40 characters.
<b>text</b>	pa_text	list	The lines of text of the part (should only be used if the text is a plain text file). There is no limit on the number of characters.
<b>text raw</b>	pa_text_raw	string	The text of the part as a single raw string. There is no limit on the number of characters.
<b>comp text</b>	pa_comp_text	list	The lines of text of the component, for use when the current record is a version. There is no limit on the number of characters.
<b>comp text raw</b>	pa_comp_text_raw	string	The text of the component as a single raw string, for use when the current record is a version. There is no limit on the number of characters.
<b>pred_ver</b>	pa_pred_ver	string	The version-id of the version's predecessor (versions only).
<b>symname</b>	pa_symb-name	string	The symbolic name of the version (versions only). Maximum 19 characters.
<b>user</b>	pa_user	string	The user who created the version (versions only).
<b>comp date</b>	pa_comp_date	date/time	The date on which the component was created (use when current record is a version).
<b>varb1--varb40</b>	pa_varb1--40	string, date/time, numeric	Arbitrary, site-specific information about the version (versions only). Varb1-2 and arb36-40 are limited to 120 characters; varb3--35 are limited to 40 characters.

A qualified reference to a field in the parts database requires one parameter: the full path of the part/ version.

Each version has its own name, obsolete, flags, locked, status, pred\_ver, symbname, user, date and varb fields; it inherits all other fields from the associated component part.

**Part Location**

The location of a subsystem or component type part defines the physical location represented by this part. The location of a *subsystem* type part should refer to an operating system *directory*. The location of a *component* type part should refer to an operating system *file*: this file is (normally) an VC file which contains the complete version history — see [Parts, Files and the Database](#).

It is important to understand that — at least in the normal case where the file corresponding to a component is a VC file holding the actual version of the file represented by each version of the component in the parts database — this file may *not* be directly manipulated by the user. For example, it cannot be printed, edited, compiled etc. Furthermore, since its contents are likely to be precious and available to multiple users/ products it will probably be located in a global area where it cannot (or should not) be altered by a user directly but only through the **AllChange** system, which will invoke appropriate VC tools as necessary. To obtain a copy of a version it must be *checked out* by a user to a *workspace*, which produces a file in a local area which may be examined, edited and so on. A part's location always refers to the global area where the VC file resides.

The location is inherited by all children of a subsystem part (i.e. the locations of the children are joined to this location). If an absolute path is given as the location field of a part, this replaces any inherited location.

If no location field is given then the name of the part is taken as the location field.

The effect of the above is that if location fields are left empty each subsystem in a part path produces a subdirectory with the same name in the corresponding operating system path and a final component produces a file with the same name at the end of the path. This greatly simplifies setting up systems provided operating system locations can be allowed to mirror the parts hierarchy and names (or vice versa) while still allowing this to be overridden with explicit locations at any stage.

[Figure 11.1](#) gives some sample part names in a system together with their location field and illustrates the operating system path which results from applying the preceding rules. The location column is blank where the location field of the part is empty. The operating system is Windows, under which the distinction between **AllChange** part names and operating system path names should be clear.

**Figure 11.1: Part locations**

Part	Location	Path
/	c:\	c:\
/sub1		c:\sub1
/sub1/file.doc		c:\sub1\file.doc
/sub1/helpfile	file.hlp	c:\sub1\file.hlp
/sub2	c:\public	c:\public
/sub2/sub3	src\doc	c:\public\src\doc
/sub2/sub4	d:\dir	d:\dir
/sub2/sub4/comp		d:\dir\comp

A part may have a *variable location*: this is indicated by having the location start with a *\$variable*. Instead of the location being fixed, this means that it varies depending on how the *\$variable* expands in the current invocation of **AllChange**. Variable locations are intended for use by Administrators in networks where paths vary from machine to machine or program instance to instance; see [Variable Locations](#).

The physical location for each part must be unique for that part (i.e. two or more parts may not represent the same physical entity).

The location field may be set on creation of a part or by altering the field.

The location field for a uses type part specifies the part that is used.

### Field References for Check-outs Database

Details of all parts which are checked out are maintained in the check-outs database, this is internally known as the *issues database* to **AllChange**.

#### Issue Fields

The check-outs logged in the issues database have a number of fields which may be used for querying/reporting.

The fields are:

Title	Field Reference	Type	Description
<b>foredit</b>	iu_foredit	truth	True/ false value specifying whether the part was checked out for edit. True if the part was checked out for edit, false otherwise
<b>part</b>	iu_part	string	The full path of the part checked out
<b>newpart</b>	iu_newpart	string	The full path of the new part that has been reserved/locked. This is blank if the part was not checked out for edit. If the part was checked out with pessimistic locking then this will be the new version reserved for when the part is checked back in. If the part was checked out with optimistic locking then the new version will be shown as the branch checked out on if any followed by %< <b>workspace name</b> > to show the workspace it is checked out to.
<b>wspcname</b>	iu_wspcname	string	The name of the workspace checked out to
<b>user</b>	iu_user	string	The user to whom the part was checked out.
<b>date</b>	iu_date	date/time	The date that the part was checked out.
<b>comment</b>	iu_comment	string	The comment giving the reason for the check out. Maximum 120 characters.
<b>optimistic</b>	iu_optimistic	truth	True/False specifying whether the part was checked out for edit with optimistic locking

A qualified reference to a field of the issues database requires two parameters: the (full path of) the version checked out, followed by the name of the workspace to which it is checked out.

### Field References for Instances Database

Details of all instances are maintained in the *instances* database.

#### Instance Fields

The instances logged in the instances database have a number of fields which may be used for querying/reporting.

The fields are:

Title	Field Reference	Type	Description
<b>part</b>	in_part	string	The full path name for the instance.
<b>user</b>	in_user	string	The user that created the instance.
<b>date</b>	in_date	date/time	The date that the instance was created.
<b>obsolete</b>	in_obsolete	truth	True/False specifying whether the instance is obsolete
<b>arb1--arb40</b>	in_arb1--40	string,	Arbitrary, site-specific information about the instance. Arb1-3 and

	date/time, numeric	arb36-40 are limited to 120 characters; arb4-35 are limited to 40 characters.
--	--------------------	---

A qualified reference to a field of the instances database requires the (full path of) the instance.

## Field References for Baselines Database

### Baseline Fields

The baselines database has a number of fields associated with each baseline taken. 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 (or sub-baselines if the baseline is a meta-baseline) that were included in that baseline: this is known as the baseline detail.

For each field described below an indication is given as to whether the information is contained in the baseline header and so occurs once per baseline, or whether it is contained in the baseline detail and so occurs once for each part/ sub-baseline baselined.

Title	Field Reference	Type	Description
<b>obsolete</b>	bl_obsolete	truth	True/ false indication of whether the baseline is obsolete (baseline header). True if the baseline is obsolete, false otherwise.
<b>lockparts</b>	bl_lockparts	truth	True/ false indication of whether parts baselined were locked (baseline header). True if the baseline locked its parts, false otherwise.
<b>name</b>	bl_name	string	The name of the baseline (baseline header). This may include any alpha numeric characters, plus the following _ - + : , . . A space character may not be used, and the name must start with a letter, digit or . Maximum 59 characters.
<b>user</b>	bl_user	string	The user who took the baseline (baseline header).
<b>date</b>	bl_date	date/time	The date on which the baseline was taken (baseline header)
<b>releasedate</b>	bl_release-date	date/time	The date on which the baseline was taken (baseline header).
<b>design</b>	bl_design	truth	True/false indication of whether the baseline is a design-baseline (baseline header). True if the baseline is a design baseline, false otherwise. This is provided for backward compatibility and may be removed at a future release. Use of bl_type instead is recommended.
<b>toppart</b>	bl_toppart	string	The toppart associated with the baseline (baseline header). The top part affected by the baseline
<b>class</b>	bl_class	string	The class of the baseline (baseline header). This may be any of the defined baseline classes as described in <a href="#">Classes</a> . The class of a baseline defines its life-cycle if any. The class of the baseline
<b>status</b>	bl_status	string	The current status of the baseline in its life-cycle (baseline header). 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.
<b>locked</b>	bl_locked	truth	The lock state of the baseline. True/ false indication of whether the baseline is locked against further change (baseline header).
<b>type</b>	bl_type	string	The type of the baseline, will be release, design or instance (baseline header).

<b>meta</b>	bl_meta	truth	Whether the baseline is a meta-baseline. True/ false indication of whether the baseline is a meta-baseline (baseline header).
<b>partname</b>	bl_partname	string	The full path of the part contained in the baseline (baseline detail).
<b>subblinname</b>	bl_subblinname	string	The name of the sub-baseline contained in the meta-baseline (baseline detail).
<b>comment</b>	bl_comment	string	The comment associated with the baseline (baseline header).
<b>text</b>	bl_text	list	The lines of text of the baseline (should only be used if the text is a plain text file). There is no limit on the number of characters.
<b>text raw</b>	bl_text_raw	string	The text of the baseline as a single raw string. There is no limit on the number of characters.
<b>blinesaffected</b>	bl_bline-saffected	list	A list of baselines that are related to the baseline as a baseline affected.
<b>filesaffected</b>	bl_file-saffected	list	A list of the files that are attached to the baseline. These work the same way as CR Attachments — see <a href="#">CR Attachments</a> .
<b>arb1–40</b>	bl_arb1–40	string, date/time, numeric	Arbitrary, site-specific information about the baseline (baseline header). Arb1–2 and arb36–40 are limited to 120 characters; arb3–35 are limited to 40 characters.
<b>detail_type</b>	bl_detail_type	string	The (nomenclature-mapped) type of the detail in a baseline. Will be one of (nomenclature-mapped): "baseline", "instance", "version" or "part".
<b>dtarb1–40</b>	bl_dtarb1–40	string, date/time, numeric	Arbitrary, site-specific information about the baseline detail. Arb1–2 and arb36–40 are limited to 120 characters; arb3–35 are limited to 40 characters.

A qualified reference to most fields in the baselines database requires one parameter: the baseline name; however, the fields which refer to a baseline detail (`bl_partname`, `bl_subblinname`, `bl_detail_type`, `bl_dtarb1-40`) require two parameters: the baseline name, followed by the name of the item in the detail (partname or sub-baseline name). About Baselines Database

The *baselines database* contains information about baselines that have been taken.

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 (or sub-baselines if the baseline is a meta-baseline) that were included in that baseline: this is known as the *baseline detail*.

### Field References for Monitors Database

The information about what events and objects are monitored by whom is stored in the *monitors database*.

The System Administrator should define the `monitor_action` entry in the commands file to perform whatever actions are required on an event being triggered. Normally this should be that the operating system's "mail" facility is used to post an appropriate message to the users.

### Monitor Fields

The monitors which have been placed and stored in the monitors database have a number of fields which may be used for querying/ reporting.

The fields are:

Title	Field Reference	Type	Description
-------	-----------------	------	-------------

<b>item</b>	mo_item	string	The name of the object which is being monitored.
<b>event</b>	mo_event	string	The event which is being monitored for this item.
<b>user</b>	mo_user	string	The user who placed the monitor.
<b>arb1</b>	mo_arb1	string, date/time, numeric	Arbitrary, site-specific information about the monitor. Maximum 120 characters.

A qualified reference to a field in the monitors database requires one parameter: the item.

### Field References for CRs Database

#### CR Fields

The CRs stored in the CR database have a number of fields which may be used for querying/ reporting.

The fields associated with each CR are:

<b>Title</b>	<b>Field Reference</b>	<b>Type</b>	<b>Description</b>
<b>number</b>	cr_number	string	Each CR is identified with a CR "number". The CR number is determined at the time the CR is created and depends on the CR class . It may contain any combination of fixed characters, a string supplied by the user and/ or a number automatically generated by the system. CR numbering formats are totally site-definable so the <b>AllChange</b> Administrator should inform users if a convention other than the default is adopted. Maximum 19 characters.
<b>obsolete</b>	cr_obsolete	truth	True/ false indication of whether the CR is obsolete. True if the CR is obsolete, false otherwise
<b>class</b>	cr_class	string	The class of the CR. This may be any of the defined classes as described in <a href="#">Classes</a> . The class of a CR defines the life-cycle of the CR.
<b>status</b>	cr_status	string	The current status of the CR. This field is set when the status of the CR is changed and is initialised to the initial status of the CR life-cycle.
<b>toppart</b>	cr_toppart	string	The toppart affected. The CR should not affect any part higher in the design tree than this part.
<b>ref</b>	cr_ref	string	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 <i>not</i> used by the system to identify the CR. Maximum 40 characters.
<b>originator</b>	cr_originator	string	Identifies the originator of the CR. This field is automatically set to the user who created the CR.
<b>date</b>	cr_date	date/time	The date on which the CR was created.
<b>assignee</b>	cr_assignee	string	The user to whom the CR is currently assigned. This field is set when the CR is assigned.
<b>summary</b>	cr_summary	string	May be used to summarise what the CR is about. May contain any text. Maximum 120 characters.
<b>text</b>	cr_text	list	The lines of text of the CR (should only be used if the text is a plain text file). There is no limit on the number of characters.
<b>text raw</b>	cr_text_raw	string	The text of the CR as a single raw string. There is no limit on the number of characters.

<b>partsaffected</b>	cr_part-saffected	list	A list of the parts (including versions) that the CR affects. Each part will be a full part path.
<b>versionssolved</b>	cr_versionsolved	list	A list of the versions that "solved" the CR. Each version will be a full part path.
<b>blinesaffected</b>	cr_bline-saffected	list	A list of the baselines that the CR affects.
<b>blinessolved</b>	cr_blinessolved	list	A list of the baselines that "solved" the CR.
<b>crsaffected</b>	cr_crsaffected	list	A list of the (other) CRs that the CR affects.
<a href="#">filesaffected</a>	cr_file-saffected	list	A list of the files that are attached to the CR. These may be either full paths or plain filenames, depending on whether the attachments are copies or links.
<b>arb1--100</b>	cr_arb1--100	string, date/time, numeric	Arbitrary, site-specific information about the CR. Arb1--5, arb11--15, arb26--30, arb41--45, arb56--60, arb71--75 and arb86--90 are limited to 120 characters; arb6--10, arb16--25, arb31--40, arb46--55, arb61--70, arb76--85, arb91--100 are limited to 40 characters.

A qualified reference to a field in the CR database requires one parameter: the CR number.

### CR Attachments

Attaching a file causes the (plain, lowercase) name of the file to be added into the list of attachments and the file to be copied to a subdirectory of the **AllChange** project directory — specifically, all of a particular CR's attachments are kept as individual files with their original (plain, lowercase) filename in a subdirectory whose name is generated by the ACCEL function `attachment_dirname()` from the CR number, which in turn is in a subdirectory named `crattach` under the **AllChange** project directory.

For example, attaching `c:\docs\Descrip.doc` to CR `RFC00001` would add an attachment named `descrip.doc` and result in the file being copied to somewhere like `z:\acproj\crattach\RFC00001\descrip.doc`. The original file copied is not altered/ deleted.

When an attachment is viewed from ACE the attachment file is copied to the user's temporary directory and the associated executable program (e.g. Word) is run on this file; on exiting the associated program and returning to **AllChange** the temporary file is not deleted.

The hard-coded logic deals only with adding and removing filenames to/ from the list of files affected (cf. parts affected etc.). All of the operations treating CR files affected as file attachments are written in the user-defined ACCEL function `filesaffected_action` (found in `utility.ac`), which is called from the actions of relevant commands (**newcr/altercr**). It is possible to tailor this code to achieve additional/ different functionality, e.g. if attachments are large they could be zipped/ unzipped before/ after transfer, or in some circumstances files affected could be for informational purposes only and not involve actual attachments.

Baseline attachments are handled in exactly the same manner as CR attachments, except that the subdirectory name is `blattach`.

### Attachments as Links

It is possible to have attachments as "links" (i.e. just a reference to the original file's location is stored instead of taking a copy to the **AllChange** project directory). The full path to the file (still lowercased) is stored in the CR's files affected list, instead of just the plain filename. The only action performed on a linked file is to allow the user to edit/ view it from the **Attachments** tab; all other actions such as detaching it or deleting the CR have *no* effect on the file.

Each individual attachment may be either a copy or a link, independently of others. The behaviour is governed by a `config.ac` option, `FileAttachmentsAsLinks`, which can have values `Never`, `Always` or `Ask`. If it is `Never` attachments are always copied; if it is `Always` attachments are always linked; if it is `Ask` (the default) then whenever a user attaches a file in ACE he will be asked whether to attach as a link or a copy.

Linked attachment support has been added in response to customer demand. Their advantage over copies is mostly space saving when attaching the same file to multiple CRs; their disadvantage is lack of security and no guarantee of permanence. If you decide to allow links, bear in mind:

- the file pointed to by a link could be deleted or changed at any time
- there is no guarantee that the path a user employs to attach a file will be correct for other users (e.g. `c:\...` is probably a bad idea)

We discourage the use of links for these reasons.

### Field References for Status Logs Database

Information about the status changes undergone by parts, baselines and/or CRs is held in the *status logs database*.

The system will automatically add a record to this database whenever the status of an item is altered through the **status**, **statusbaseline** or **statuscr** commands; it will also add a record for the initial status when an item is first created, and one when a CR is assigned. All status logs for an item are deleted if the item is deleted; all status logs for a part (and its versions, if any) are copied/renamed if the part is copied/renamed.

Status logging can be enabled separately for each of the three databases. By default status logging is enabled for all three databases. A site may disable status logging on any database to save storage space. Alternatively automatic status logging may be disabled and the command definitions and life-cycle files may instead be configured to explicitly create logs of selected events only.

The status logs database may be used for querying/reporting purposes. Although there is only really one status log database shared for logs of parts, baselines and CRs, reports provide access to the status log database separately for each type of item logged.

### Status Log Fields

The fields associated with each status log are:

Title	Field Reference	Type	Description
<b>status_log_item</b>	??_status_log_item	string	The item whose status change is being logged. The item may be a partname ( subsystem , component and version), baseline name or CR number.
<b>status_log_class</b>	??_status_log_class	string	The item's class when logged.
<b>status_log_status</b>	??_status_log_status	string	The item's new status. This will be set to <code>Assigned</code> in the case of a CR assignment.
<b>status_log_date</b>	??_status_log_date	date/time	The date/ time of the status change
<b>status_log_user</b>	??_status_log_user	string	The user making the change
<b>status_log_arb1</b>	??_status_log_arb1	string,	Arbitrary, site-specific information about the status change log. Maximum 119 char-

<b>_log_arb1</b>	date/time, numeric	acters. This is set to the user to whom a CR is assigned in the case of a CR assignment.
------------------	--------------------	--

Although there is only really one status log database shared for logs of parts, baselines and CRs, ACCEL provides access to the status log database separately for each type of item logged. The names of these databases are `part_status_log`, `baseline_status_log` and `cr_status_log` respectively; this is used in reports and for the ACCEL `each` statement. The field reference prefix is the standard one for the type of item, e.g. the name of the item logged for the parts, baselines and CRs databases is accessed from ACCEL as `pa_status_log_item`, `bl_status_log_item` and `cr_status_log_item` respectively. The status log database does not support qualified references.

### Field References for Items Affected Database

Information about the relationships between parts, baselines and CRs and the parts, baselines and crs they relate to is held in the *items affected database*.

The relationships can be enabled separately for each of the three databases. By default all relationships are enabled for all three databases. A site may disable relationships if they unused in order to simplify the system and the user interface.

The items affected database may be used for querying/reporting purposes. Although there is only really one items affected database shared for all relationships of parts, baselines and CRs, reports provide access to the items affected database separately for each type of item to which there are relationships.

### Items Affected Fields

The fields associated with each item affected are:

Title	Field Reference	Type	Description
<b>itemaffecting</b>	??ia_ite-maffecting	string	The item which has the relationship to other items.
<b>itemaffectingtype</b>	??ia_ite-maffectingtype	string	The type of the item affecting, this may be cr, part, baseline or file
<b>itemsaffected</b>	??ia_ite-maffected	string	The item which is related to the itemaffecting
<b>itemaffectedtype</b>	??ia_ite-maffectedtype	string	The type of the item affected, this may be cr, part, baseline or file
<b>solved</b>	??ia_solved	truth	whether this is an affected or solved relationship
<b>date</b>	??ia_date	date	The date when this relationship was added
<b>user</b>	??ia_user	string	The user who added this relationship
<b>class</b>	??ia_class	string	The class the item had at the time the relationship was added
<b>arb1-40</b>	??ia_arb1-40	string	Arbitrary fields 1-40 for the item affecting relationship

Although there is only really one items affected database shared for parts, baselines and CRs, ACCEL provides access to the items affected database separately for each type of item. The names of these databases are `part_items_affected`, `cr_items_affected` and `baseline_items_affected` respectively; this is used in reports and for the ACCEL `each` statement. The field reference prefix is the standard one for the type of item, e.g. the name of the item logged for the parts, baselines and CRs databases is accessed from ACCEL as `paia_itemaffecting`, `blia_itemaffecting` and `cria_ite-maffecting` respectively. The items affected database requires 4 parameters for a qualified reference: item affecting, item affected, item affected type and whether solved or not e.g. `cria_date('CR00001, /doc1.txt;002', 'part', true)`.

## Field References for Votes Database

Information about the votes cast for parts, baselines and CRs is held in the *votes database*.

The system will automatically add a record to this database whenever a vote is cast and whenever a status is entered which starts a vote. All votes for an item are deleted if the item is deleted; all votes for an item are copied/renamed if the item is copied/renamed.

Voting can be enabled separately for each of the three databases. By default voting is enabled for all three databases.

The votes database may be used for querying/reporting purposes. Although there is only really one votes database shared for parts, baselines and CRs, reports provide access to the votes database separately for each type of item.

### Vote Fields

The fields associated with each vote are:

Title	Field Reference	Type	Description
<b>item</b>	??vt_item	string	The item voted on (part, cr or baseline)
<b>status</b>	??vt_status	string	The status of the item voted on
<b>user</b>	??vt_user	string	The user that cast the vote or caused the vote to be opened or closed
<b>representing</b>	??vt_representing	string	The role that the user selected to cast the vote as representing
<b>date</b>	??vt_date	date/time	The date and time that the vote was cast/opened/closed
<b>vote</b>	??vt_vote	string	The status that was voted for, or (Vote Start) or (Vote End)
<b>comment</b>	??vt_comment	string	Any comment that was specified with the vote
<b>arb1–40</b>	??vtarb1–40	string, date/time, numeric	Arbitrary, site-specific information about the vote. Arb1–5, arb11–15 and arb26–30 are limited to 120 characters; arb6–10, arb16–25 and arb31–40 are limited to 40 characters.

Although there is only really one votes database shared for votes on parts, baselines and CRs, ACCEL provides access to the votes database separately for each type of item. The names of these databases are *part\_vote*, *baseline\_vote* and *cr\_vote* respectively; this is used in reports and for the ACCEL *each* statement. The field reference prefix is the standard one for the type of item, e.g. the item voted for the parts, baselines and CRs databases is accessed from ACCEL as *pavt\_item*, *blvt\_item* and *crvt\_item* respectively.

A qualified reference requires 3 parameter, item, date and user e.g. `crvt_comment('CR00001', '2008/10/07 21:10:45', 'paul')`.

## Field References for Pools Database

Title	Field Reference	Type	Description
<b>name</b>	po_name	string	The name of the pool
<b>dir</b>	po_dir	string	The directory for the pool
<b>serverdir</b>	po_serverdir	string	The directory for the pool from the server's point of view
<b>ftphost</b>	po_ftphost	string	The name of the host machine for an FTP pool
<b>ftpos</b>	po_ftpos	string	The name of the host

			operating system for an FTP pool
<b>ftppath</b>	po_ftppath	string	The path on the host operating system for an FTP pool

A qualified reference to a field in the pools database requires one parameter: the pool name. Further details on the fields of the pools database may be found in [Pools](#).

### Field References for Classes Database

Title	Field Reference	Type	Description
<b>classname</b>	cl_classname	string	The name of the class
<b>classtype</b>	cl_classtype	string	The type of item for the class
<b>cyclename</b>	cl_cyclename	string	The name of the cycle for the class
<b>cr_num_fmt</b>	cl_cr_num_fmt	string	The CR numbering format to be used for the class
<b>text_template</b>	cl_text_template	string	The name of the template file to be used for the text field of items of this class.
<b>ver_template</b>	cl_ver_template	string	The name of the template file to be used for version text fields of parts of this class. This is valid for component class types only.
<b>part_template</b>	cl_part_template	string	The name of the template file used as the contents of the initial version of a part of this class. This is valid for component class types only.
<b>classdefault</b>	cl_classdefault	truth	true if this class is the default for its class type, false otherwise
<b>attributes</b>	cl_attributes	list	The attributes of the class
<b>description</b>	cl_description	string	The description/tool tip for this class

The following fields should no longer be used, and exist only for backward compatibility:

<b>cr_tpl</b>	cl_cr_tpl	string	Superceded by cl_text_template
<b>template</b>	cl_template	string	Superceded by cl_text_template/cl_part_template

A qualified reference to a field in the classes database requires one parameter: the class name. Further details on the fields of the classes database may be found in [Classes](#).

### Field References for Workspaces Database

Title	Field Reference	Type	Description
<b>wrkname</b>	ws_wrkname	string	The name of the workspace
<b>wrkdir</b>	ws_wrkdir	string	The directory associated with the workspace
<b>users</b>	ws_users	list	The users permitted to use the workspace
<b>partname</b>	ws_partname	string	The part associated with the workspace
<b>hierarchical</b>	ws_hierarchical	truth	true if the workspace is hierarchical, false otherwise
<b>autoupdate</b>	ws_autoupdate	truth	true if the workspace is to be updated automatically, false otherwise
<b>poollist</b>	ws_poollist	list	The pools currently registered for the workspace
<b>registrations</b>	ws_registrations	list	The part registrations in effect for the workspace
<b>webworkspace</b>	ws_web-	truth	true if the workspace is a Web workspace

	workspace		
<b>weballowinace</b>	ws_weballowinace	truth	true if a Web workspace may also be used in ACE
<b>webtransfer</b>	ws_weballowinace	truth	true if a Web workspace allows files to be up/downloaded
<b>downloaddir</b>	ws_downloaddir	string	The download directory for a Web workspace
<b>ftphost</b>	ws_ftphost	string	The name of the host machine for an FTP workspace
<b>ftpos</b>	ws_ftpos	string	The name of the host operating system for an FTP workspace
<b>ftppath</b>	ws_ftppath	string	The path on the host operating system for an FTP workspace
<b>webhost</b>	ws_webhost	string	The name of the host machine for an FTP Web deployment workspace
<b>webos</b>	ws_webos	string	The name of the host operating system for an FTP Web deployment workspace
<b>webpath</b>	ws_webpath	string	The path on the host operating system for an FTP Web deployment workspace

A qualified reference to a field in the workspaces database requires one parameter: the workspace name. Further details on the fields of the workspaces database may be found in [Workspaces](#).

#### Field References for Roles Database

Title	Field Reference	Type	Description
<b>partname</b>	ro_partname	string	The part associated with the role
<b>role</b>	ro_role	string	The name of the role
<b>users</b>	ro_users	list	The users who have the role

A qualified reference to a field in the roles database requires one parameter: the role name. Further details on the fields of the roles database may be found in [Role Definitions](#).

#### Field References for Cycles Database

Title	Field Reference	Type	Description
<b>name</b>	cy_name	string	The name of the cycle
<b>cycle type</b>	cy_cycle_type	string	The type of the cycle (what kind of item it can be applied to)
<b>status name</b>	cy_status_name	string	The name of a status
<b>attributes</b>	cy_attributes	list	The attributes of the status
<b>progressions</b>	cy_progressions	list	The statuses to which you can progress from the status
<b>defprogression</b>	cy_defprogression	string	Returns the default progression. Qualified reference requires a cycle and status name.
<b>newver</b>	cy_newver	truth	true if entering the status causes a new version of the component to be created, false otherwise
<b>status type</b>	cy_status_type	string	The type of a status ( <i>open</i> or <i>closed</i> )
<b>entry cond</b>	cy_entry_cond	list	The entry condition for the status
<b>entry action</b>	cy_entry_action	list	The entry action for the status
<b>exit cond</b>	cy_exit_cond	list	The exit condition for the status
<b>exit action</b>	cy_exit_action	list	The exit action for the status

A qualified reference using `cy_name` or `cy_cycle_type` requires one parameter: the cycle name; all other fields of the cycles database require two parameters: the cycle name, followed by the status name. Further details on the fields of the cycles database may be found in [Life-cycles](#).

#### Field References for Command Definitions Database

Title	Field Reference	Type	Description
<b>command name</b>	<code>fm_command_name</code>	string	The name of the command
<b>entry cond</b>	<code>fm_entry_cond</code>	list	The entry condition for the command
<b>command</b>	<code>fm_command</code>	list	The commands to be issued

A qualified reference to a field in the command definitions database requires one parameter: the command name. Further details on the fields of the command definitions database may be found in [Command Definitions](#).

#### Field References for Fieldnames Database

Title	Field Reference	Type	Description
<b>oldname</b>	<code>fl_oldname</code>	string	Old/ real arbitrary field name
<b>newname</b>	<code>fl_newname</code>	string	New arbitrary field name
<b>displayname</b>	<code>fl_displayname</code>	string	Displayed arbitrary field name
<b>listtext</b>	<code>fl_listtext</code>	list	ACCEL expression for possible values
<b>liststrings</b>	<code>fl_liststr</code>	list	Literal list of possible values
<b>entrytypes</b>	<code>fl_entrytype</code>	string	Type of control for entering data
<b>editable</b>	<code>fl_editable</code>	truth	Is field user-editable?
<b>datatype</b>	<code>fl_datatype</code>	string	Field data type
<b>compulsory</b>	<code>fl_compulsory</code>	truth	Is field compulsory?
<b>classes</b>	<code>fl_classes</code>	list	Classes to which this arbitrary field pertains
<b>helpstring</b>	<code>fl_helpstring</code>	string	The help string for the field
<b>length</b>	<code>fl_length</code>	numeric	The maximum number of characters allowed in the field
<b>tabname</b>	<code>fl_tabname</code>	string	The name of the tab on which the field is shown
<b>span</b>	<code>fl_span</code>	string	The number of columns the field occupies on the tab
<b>check_spelling</b>	<code>fl_check_spelling</code>	truth	Whether spell checking is enabled for the field. Only applies to 'Fillin' fields
<b>attributes</b>	<code>fl_attributes</code>	list	The attributes for this field

A qualified reference to a field in the fieldname database requires two parameters: the old fieldname and the class . Further details on the fields of the fieldnames database may be found in [Field Name Definitions](#).

#### Field References for Field Tabs Database

Title	Field Reference	Type	Description
<b>name</b>	<code>ft_name</code>	string	The name of the tab
<b>database</b>	<code>ft_database</code>	string	The database to which the tab applies
<b>classes</b>	<code>ft_classes</code>	list	The classes for which this tab is defined
<b>fields</b>	<code>ft_fields</code>	list	The fields which are shown on this tab

A qualified reference to a field in the field tab database requires two parameters: the database name and the tab name. Further details on the fields of the field tabs database may be found in [Field Tabs](#).

#### Field References for Rolemap Database

<b>override</b>	<code>rm_override</code>	truth	Overrides other command access checks?
-----------------	--------------------------	-------	--

<b>command</b>	rm_command	string	Command name
<b>classes</b>	rm_classes	list	Classes to which this command access applies
<b>class not equal</b>	rm_class_not_equal	truth	Command access applies when <i>not</i> of class
<b>option</b>	rm_option	string	Option to which this command access applies
<b>option not equal</b>	rm_option_not_equal	truth	Command access applies when option <i>not</i> used
<b>cycle</b>	rm_cycle	string	Cycle to which this command access applies
<b>status</b>	rm_status	string	Status to which this command access applies
<b>roles</b>	rm_roles	list	Roles required to be allowed command access
<b>override</b>	rm_override	truth	Overrides other command access checks?

A qualified reference to a field in the rolemap database is not permitted. Further details on the fields of the rolemap database may be found in [Command Access](#).

#### Field References for Configuration Options Database

Title	Field Reference	Type	Description
<b>name</b>	cf_name	string	Configuration option name
<b>value</b>	cf_value	string	Configuration option value
<b>unknown</b>	cf_unknown	string	User defined (not built-in) option

A qualified reference to a field in the configuration options database requires one parameter: the name. Further details on the fields of the configuration options database may be found in [Configuration Options](#).

#### Field References for Includes Database

Title	Field Reference	Type	Description
<b>filename</b>	ic_filename	string	Included function filename
<b>comment</b>	ic_comment	string	Included file comment
<b>in use</b>	ic_in_use	truth	Is this file in use?
<b>compulsory</b>	ic_compulsory	truth	Is this file compulsory?

A qualified reference to a field in the includes database requires one parameter: the name. Further details on the fields of the includes database may be found in [Function Definition Files](#).

#### Field References for Report Formats Database

Title	Field Reference	Type	Description
<b>filename</b>	rf_filename	string	Report format filename
<b>category</b>	rf_category	string	Report format file category
<b>description</b>	rf_description	string	Report format file description
<b>location</b>	rf_location	string	Report format file logical location
<b>disabled</b>	rf_disabled	truth	Is report format file disabled?
<b>access</b>	rf_access	string	Who can access this report format file
<b>users</b>	rf_users	list	Enumerated list of report format file users

A qualified reference to a field in the report formats database requires one parameter: the filename. Further details on the fields of the includes database may be found in [Report Formats](#).

#### Field References for User Registrations Database

Title	Field Reference	Type	Description
<b>name</b>	us_name	string	User name
<b>mailname</b>	us_mailname	string	User's mail name
<b>fullname</b>	us_fullname	string	The full name specified for the user. If no full name is

			specified, then it will be the user name (us_name)
<b>ftpnames</b>	us_ftpnames	list	User's FTP names (list of items, each of which is <i>remote-machine-name</i> <sup>^</sup> <i>remote-username</i> ) where <sup>^</sup> <i>t</i> is a tab character.
<b>admin</b>	us_admin	truth	Is user an <b>AllChange</b> administrative user?
<b>cr only</b>	us_cr_only	truth	Is user an <b>AllChange</b> CR-only user?
<b>obsolete</b>	us_obsolete	truth	Is user an obsolete <b>AllChange</b> user. i.e. are they no longer an AllChange user
<b>project</b>	us_project	truth	Is the user a valid user for the current project

A qualified reference to a field in the user registration database requires one parameter: the user name. Further details on the fields of the includes database may be found in [User Registration](#).

### Field References for Groups Database

Title	Field Reference	Type	Description
<b>name</b>	gp_name	string	Group name
<b>mailname</b>	gp_mailname	string	Mail name for group
<b>users</b>	gp_users	list	Users who are members of this group
<b>location</b>	gp_location	string	The directory containing the group definitions

A qualified reference to a field in the group database requires one parameter: the group name. Further details on the fields of the includes database may be found in [User Groups](#).

### Field References for Nomenclature Mapping Database

Title	Field Reference	Type	Description
<b>oldname</b>	nm_oldname	string	The default (Intasoft) name
<b>newname</b>	nm_newname	string	The replacement name
<b>plural</b>	nm_plural	string	The pluralised form of the new name
<b>apostrophised</b>	nm_apostrophised	string	The apostrophised form of the new name

A qualified reference to the nomenclature database requires one parameter: the old name, e.g. `nm_plural('cr')`. Further details on the fields of the nomenclature database may be found in [Nomenclature Mapping](#)

### Field References for Branches Database

Field References for Branches Database

Title	Field Reference	Type	Description
<b>name</b>	br_name	string	The name of the branch
<b>part</b>	br_part	string	The top part associated with the branch name

A qualified reference to a field in the branches database requires one parameter: the name. Further details on the fields of the branches database may be found in [Branch Names](#).

### Field References for Part Permissions Database

Field References for Part Permissions Database

<b>recursive</b>	pp_recursive	string	Whether the permission applies to children of the part specified
------------------	--------------	--------	--

<b>part</b>	pp_part	string	The part to which the permission applies
<b>classes</b>	pp_classes	string	The classes of part to which the permission applies
<b>class_not_equal</b>	pp_class_not_equal	string	Whether the classes specified are the classes <i>not</i> to match
<b>usertype</b>	pp_usertype	string	The User Type to which the permission applies
<b>roles</b>	pp_roles	string	The Roles which have permission to view the parts
<b>recursive</b>	pp_recursive	string	Whether the permission applies to children of the part specified

A qualified reference to a field in the part permissions database requires two parameters: the part and the class. Further details on the fields of the part permissions database may be found in Edit Part Read-Permissions.

## Field References for the Vote Definitions Database

Title	Field Reference	Type	Description
<b>type</b>	vd_type	string	The database to which the votes apply
<b>cycle</b>	vd_cycle	string	the life cycle for which the vote is defined
<b>status</b>	vd_status	string	the status for which the vote is defined
<b>deadline</b>	vd_deadline_field	string	the name of the field used to calculate the deadline (if any). (Vote Start Date) may be used to reflect the date the vote started/was opened. i.e. the date it entered the voting status and the (Start Vote) entry was created in the votes database. This may be the Display Name for any date based arbitrary field or the name of a built in date field
<b>deadline offset</b>	vd_deadline_offset	numeric	the number of days to be added to the deadline field value to calculate the deadline
<b>voters</b>	vd_voters	string	a list of the voters defined for the vote. For each voter tab separated information is: voter, type, mandatory,group votes required,group votes allowed
<b>serial</b>	vd_serial	truth	Whether the vote is a serial vote
<b>criteria</b>	vd_criteria	string	serial vote stop criteria. This is an ACCEL expression
<b>decision type</b>	vd_decision_type	string	The decision type for the votes.
<b>num votes required</b>	vd_num_required	numeric	The number of votes required if the decision type is <b>Specified number</b>
<b>auto progress</b>	vd_auto_progress	truth	whether the item should be automatically progressed when a decision is reached
<b>next word</b>	vd_next_word	string	the word to display to the user to pass a serial vote to the next voter
<b>stop word</b>	vd_stop_word	string	the word to display to the user to stop a serial vote
<b>mail initial subject</b>	vd_mail_initial_subject	string	the subject of the email sent to voters on commencing a vote. If first character is & then an ACCEL expression follows
<b>mail initial text</b>	vd_mail_intial_body	string	the body of the email sent to voters on commencing a vote. If first character is & then an ACCEL expression follows
<b>mail decision type</b>	vd_mail_decision_type	string	the type of the mail recipient (user, group or field) for the email sent to voters when a decision is reached
<b>mail decision recipient</b>	vd_mail_decision_recipient	string	the recipient of the email sent to voters when a decision is reached. This may be a user, group or field name depending on the <b>mail decision type</b>
<b>mail decision subject</b>	vd_mail_decision_subject	string	the subject of the email sent to voters on a decision being reached. If first character is & then an ACCEL expression follows
<b>mail decision text</b>	vd_mail_decision_body	string	the body of the email sent to voters on a decision being reached. If first character is & then an ACCEL expression follows
<b>mail blocked type</b>	vd_mail_blocked_type	string	the type of the mail recipient (user, group or field) for the email sent to voters when a vote is

<b>mail blocked recipient</b>	vd_mail_blocked_recipient	string	blocked the recipient of the email sent to voters when a decision is blocked. This may be a user, group or field name depending on the <b>mail blocked type</b>
<b>mail blocked subject</b>	vd_mail_blocked_subject	string	the subject of the email sent to voters when a decision is blocked. If first character is & then an ACCEL expression follows
<b>mail blocked body</b>	vd_mail_blocked_body	string	the body of the email sent to voters when a decision is blocked. If first character is & then an ACCEL expression follows

A qualified reference to a field in the vote definitions database requires two parameters: the cycle and the status. Further details on the fields of the vote definitions database may be found in [Vote Definitions](#).

## ACCEL Functions

### About ACCEL Built In Functions

The built in ACCEL functions provide hooks to the **AllChange** system and allow operating system calls to be made.

Formally a function call has the general form:

```
<function> ::= <func-name> ( [<parms>] )
<parms>    ::= <factor> { , <factor> }
```

Each function has a name and is followed by the parameters to the function, if any, inside parentheses. Note that the parentheses must be present even if there are no parameters.

The parameters to the function are separated by a comma ( , ) and may be any of those described in [ACCEL Conditions](#).

Note that there are a number of GUI based functions allowing interaction with the user, care should be taken when using these that they are supported for both the Windows interface and the browser interface if both GUI's are to be used.

Functions are listed later on in alphabetical order and are also listed in broad [categories](#) of the likely uses of functions; note that some functions span more than one category.

### List of ACCEL Built In Functions by Category

#### ACCEL Functions (inc. User Defined Functions and Variables)

[accel\\_profile](#)  
[called\\_from](#)  
[local](#)  
[trace\\_echo](#)

[accel\\_warnings](#)  
[call\\_stack](#)  
[setvar](#)  
[trace\\_echo\\_map](#)

[appendlistvar](#)  
[defined](#)  
[setvar\\_list](#)  
[var](#)

[append](#)  
[evaluat](#)  
[show\\_v](#)  
[wild\\_va](#)

#### Access Control Functions

[apply\\_read\\_permissions](#)  
[has\\_perm](#)

[check\\_rolemap](#)  
[has\\_role](#)

[check\\_rolemap\\_user](#)  
[is\\_field\\_readable](#)

[clear\\_s](#)  
[role\\_us](#)

#### AllChange Command Line

[getarg](#)

[getarg\\_arb](#)

[interpret](#)

[is\\_com](#)

**Arbitrary Field Functions**[arbitrary\\_field\\_name](#)[arbitrary\\_field\\_new\\_values](#)[arbitrary\\_field\\_oldname](#)[arbitran](#)**Baseline Functions**[baseline\\_basename](#)[baseline\\_template\\_text](#)[baseline\\_text\\_filename](#)[baselin](#)[baseline\\_version](#)[baselined\\_version](#)[blinesaffecting\\_bline](#)[blinesa](#)**Configuration File Functions**[classes\\_of\\_type](#)[class\\_default](#)[full\\_to\\_username](#)[full\\_to\\_usenamelist](#)[is\\_group\\_user](#)[is\\_in\\_group\\_list](#)[is\\_project\\_user](#)[option](#)[read\\_branches](#)[read\\_functions](#)[read\\_lockouts](#)[read\\_pools](#)[read\\_rolemap](#)[read\\_workspaces](#)[status\\_is\\_open](#)[status\\_list](#)[user\\_to\\_fullnamelist.htm](#)**CR Functions**[crsaffecting](#)[crsaffecting\\_bline](#)[crsaffecting\\_cr](#)[crsaffect](#)[crsolved\\_by\\_bline](#)[cr\\_part\\_versionsolved](#)[cr\\_template\\_text](#)[cr\\_text](#)[cr\\_text\\_section\\_list](#)[is\\_in\\_cr\\_partsaffected](#)[renamecr](#)**Date/Time Manipulation Functions**[conv\\_year4](#)[date\\_compare](#)[date\\_to\\_days](#)[date\\_to](#)[format\\_date](#)[getservertime](#)[gettime](#)[localtim](#)[sys\\_date](#)[unformat\\_date](#)[utc\\_to\\_localtime](#)**Error Handling Functions**[abort](#)[error](#)[error\\_map](#)**File/Directory Functions (Client)**[attachment\\_dimame](#)[bt\\_filename](#)[cmpfiles](#)[copyfile](#)[dimame](#)[dir\\_exists](#)[dir\\_size](#)[fclose](#)[file\\_exists](#)[file\\_locked](#)[file\\_suffix](#)[file\\_time](#)[flock](#)[fopen](#)[freadline](#)[fwriteline](#)[plain\\_filename](#)[protfile](#)[renfile](#)[set\\_file\\_time](#)[update\\_bt\\_file](#)[wild\\_dirs](#)[wild\\_files](#)[writefile](#)**File/Directory Functions (Server)**[copyserverfile](#)[createdir](#)[createserverdir](#)[delserverdir](#)[file\\_exists\\_in\\_projdir](#)[file\\_exists\\_in\\_sysdir](#)[file\\_exists\\_in\\_templatedir](#)[fopen\\_in\\_projdir](#)[fopen\\_in\\_templatedir](#)[getserverfile](#)[protserverfile](#)[putserverfile](#)[serverdir\\_exists](#)[serverdir\\_size](#)[serverfile\\_exists](#)[serverfile\\_size](#)[serverfile\\_writable](#)[server\\_env\\_var](#)[server\\_oscommand](#)[server\\_set\\_env\\_var](#)[temp\\_dimame](#)[temp\\_filename](#)[unprotserverfile](#)[wild\\_server\\_dirs](#)**Generic Database Functions**[db\\_tran\\_flush](#)[db\\_tran\\_flush\\_no\\_backout](#)[display\\_index\\_names](#)[display\\_index\\_to\\_rea](#)[echo\\_log\\_error](#)[get\\_cnum\\_resource\\_value](#)[get\\_next\\_item](#)[get\\_prev\\_item](#)[make\\_user\\_list](#)[num\\_records](#)[num\\_records\\_in\\_range](#)[set\\_cnum\\_resource](#)[status\\_list](#)

**GUI functions without HTML support**

<a href="#">activate_app</a>	<a href="#">app_minimise</a>	<a href="#">app_restore</a>	<a href="#">check_</a>
<a href="#">copy_clipboard</a>	<a href="#">cut_clipboard</a>	<a href="#">deactivate_app</a>	<a href="#">default</a>
<a href="#">file_browser_reread</a>	<a href="#">get_part_filter</a>	<a href="#">get_selected</a>	<a href="#">get_sel</a>
<a href="#">get_selected_list_db</a>	<a href="#">get_selected_part</a>	<a href="#">get_upfront_window</a>	<a href="#">is_app</a>
<a href="#">paste_clipboard</a>	<a href="#">paste_string</a>	<a href="#">prompt_command</a>	<a href="#">prompt</a>
<a href="#">show_floating_output</a>	<a href="#">show_floating_progress</a>	<a href="#">show_output</a>	<a href="#">show_v</a>

**GUI functions with HTML support**

<a href="#">do_dialog</a>	<a href="#">echo</a>	<a href="#">echo_map</a>	<a href="#">end_progressbar</a>
<a href="#">get_window_field</a>	<a href="#">get_window_item</a>	<a href="#">message_box</a>	<a href="#">prompt</a>
<a href="#">prompt_blineversionlist</a>	<a href="#">prompt_condedit</a>	<a href="#">prompt_crlst</a>	<a href="#">prompt_dirlist</a>
<a href="#">prompt_filelist</a>	<a href="#">prompt_formatlist</a>	<a href="#">prompt_instancelist</a>	<a href="#">prompt_list</a>
<a href="#">prompt_partlist</a>	<a href="#">prompt_subsyslist</a>	<a href="#">prompt_userlist</a>	<a href="#">prompt_versionlist</a>
<a href="#">update_statusbar</a>			

**Instance Functions**

<a href="#">is_instance</a>	<a href="#">instance_part</a>	<a href="#">instance_list</a>	<a href="#">instance_part</a>
<a href="#">part_name_instance</a>	<a href="#">part_version_instance</a>	<a href="#">top_instance</a>	

**Monitor Functions**

[check\\_monitor](#)

**Operating System Functions**

<a href="#">call_dde</a>	<a href="#">call_dll</a>	<a href="#">call_ole</a>	<a href="#">command</a>
<a href="#">env_var</a>	<a href="#">find_executable</a>	<a href="#">ini_var</a>	<a href="#">reg_key_value_str</a>
<a href="#">set_ini_var</a>	<a href="#">shell_execute</a>	<a href="#">shell_refile</a>	<a href="#">shell_show_file</a>
<a href="#">socket_accept</a>	<a href="#">socket_close</a>	<a href="#">socket_connect</a>	<a href="#">socket_file</a>

**Part Functions**

<a href="#">calc_newversion</a>	<a href="#">check_valid_branch_name</a>	<a href="#">child_list</a>	<a href="#">def_ver</a>
<a href="#">is_def_version</a>	<a href="#">is_in_baseline</a>	<a href="#">is_in_or_partsaffected</a>	<a href="#">is_issu</a>
<a href="#">is_version</a>	<a href="#">issued_version</a>	<a href="#">partsaffected</a>	<a href="#">partsaf</a>
<a href="#">part_filename</a>	<a href="#">part_name</a>	<a href="#">part_parent</a>	<a href="#">part_te</a>
<a href="#">part_text_section</a>	<a href="#">part_text_section_list</a>	<a href="#">part_version</a>	<a href="#">part_ve</a>
<a href="#">renameversion</a>	<a href="#">version_list</a>	<a href="#">version_successor</a>	<a href="#">version</a>

**Path Manipulation Functions**

<a href="#">conv_path</a>	<a href="#">join_instance</a>	<a href="#">join_paths</a>	<a href="#">join_parts</a>
<a href="#">join_version</a>	<a href="#">make_bt_file</a>	<a href="#">relative_part</a>	<a href="#">relative_path</a>

**Reporting**

<a href="#">pagenumber</a>	<a href="#">repcommand</a>	<a href="#">report_format_list</a>	<a href="#">rep_str</a>
----------------------------	----------------------------	------------------------------------	-------------------------

**Status Log Functions**

<a href="#">add_status_log</a>	<a href="#">alter_status_log</a>	<a href="#">del_status_log</a>
--------------------------------	----------------------------------	--------------------------------

**String/List Manipulation**

<a href="#">extract_token</a>	<a href="#">format_number</a>	<a href="#">instr</a>	<a href="#">is_in_li</a>
-------------------------------	-------------------------------	-----------------------	--------------------------

<a href="#">join</a>	<a href="#">joinquote</a>	<a href="#">listlen</a>	<a href="#">lowerca</a>
<a href="#">match_wild</a>	<a href="#">midlist</a>	<a href="#">midstr</a>	<a href="#">multisu</a>
<a href="#">quote</a>	<a href="#">quote_accel</a>	<a href="#">quote_accellist</a>	<a href="#">quote a</a>
<a href="#">quote_os</a>	<a href="#">quote_oslist</a>	<a href="#">rinstr</a>	<a href="#">sortlist</a>
<a href="#">split</a>	<a href="#">splitquote</a>	<a href="#">splitunquote</a>	<a href="#">strlen</a>
<a href="#">trim_spaces</a>	<a href="#">unquote</a>	<a href="#">unquotelist</a>	<a href="#">unquote</a>
<a href="#">unquote_os</a>	<a href="#">unquote_oslist</a>	<a href="#">uppercase</a>	<a href="#">usereql</a>

**Workspace/Pool Functions**

<a href="#">attach_pseudo_workspace</a>	<a href="#">is_workspace_user</a>	<a href="#">part_pool_filename</a>	<a href="#">part_w</a>
<a href="#">part_workspace_filename</a>	<a href="#">work_filename_part</a>		

**VC File Functions**

<a href="#">control</a>	<a href="#">copyvcfile</a>	<a href="#">createvmdir</a>	<a href="#">delvcdi</a>
<a href="#">putaway</a>	<a href="#">renvcfile</a>	<a href="#">set_putaway_cmd_result</a>	<a href="#">takeout</a>
<a href="#">vccommand</a>	<a href="#">vmdir_exists</a>	<a href="#">vcerase</a>	<a href="#">vcfile e</a>
<a href="#">vcrep</a>	<a href="#">vcscan</a>		

**Voting Functions**

<a href="#">can_pass_vote_next</a>	<a href="#">check_open_votes</a>	<a href="#">current_vote_users</a>	<a href="#">get_firs</a>
<a href="#">is_open_vote</a>	<a href="#">is_voter_mandatory</a>	<a href="#">is_voter_optional</a>	<a href="#">num_vo</a>
<a href="#">valid_voters</a>	<a href="#">vote_decision</a>	<a href="#">vote_deadline</a>	<a href="#">votes c</a>
<a href="#">vote_start_date</a>	<a href="#">votes_cast</a>	<a href="#">vote_initial_mail_users</a>	

**A–Z list of ACCEL Built In Functions*****abort***

`abort(message)`

Causes a fatal error to be raised similar to `error()`. The difference is that `abort()` aborts all operations, whereas `error()` aborts only the current operation and continues to the next. This distinction is important where an operation is performed on multiple items.

As an example, take writing the entry condition for deleting a part. If the user selects multiple parts this will be called for each item in turn. If the user is not allowed to delete a particular part — say as a result of a `has_perm()` check failing — use `error()` so that execution proceeds to trying to delete the next part. But if the user is not allowed to delete any part — say just by a test on his username — use `abort()` so that no attempt is made to delete any further parts.

**Example**

```
if call(IsCycleDriven, pa_class) and # Has a cycle
  not has_perm(pa_part , 'dbadmin') then # dbadmin may bypass cycle driven
  # enforce life-cycle driven
  if not called_from('status') then
    abort(map_nomenclature('Must use #status# command to #issue# #part#'));
  endif;
endif;
```

If put into the issue entry in `commands.ac` will abort an issue command if the component being issued is cycle driven and issue is not invoked from the status command

## *accel\_profile*

`accel_profile(action)`

Activates ACCEL profiling — gathering timing statistics on ACCEL function calls. *Action* may be one of the following:

<b>on</b>	start profiling
<b>off</b>	stop profiling
<b>reset</b>	reset all profiling statistics to 0
<b>show</b>	output profiling statistics

The statistics show, for every ACCEL function (in-built and user-defined) called while profiling is on: function name, number of times called, total time in calls, average time per call. This information may be used to pinpoint slow areas of code and then improve them.

## *accel\_warnings*

`accel_warnings(level)`

Determines whether ACCEL issues a warning message when certain dubious operations are performed in the language. *Level* is a number: this is treated as a bit pattern, with different bits controlling different checks. The bits currently defined are:

1	warn on an attempt to access the value of an undefined user variable (via the <code>var()</code> function)
128	treat any warnings as errors

Setting *level* to 0 switches these warnings off. Useful for debugging.

## *activate\_app*

`activate_app()`

Activates the ACE application, i.e. brings it to the foreground. This may be useful when ACE is acting as a server to some other application and is about to show the user something or require interaction.

## *add\_status\_log*

`add_status_log(itemtype, item, class, status, date, user, arb1)`

Adds a log to the status log database. *itemtype* should be one of `part`, `baseline` or `cr`. *item* should be an item appropriate to the database, e.g. `/subsys/comp1;004,blinel` or `01111` respectively. *class* and *status* should be the current class and status of the item. If *date* and/ or *user* are empty ( ' ' ) the current date/ time and/ or username will be filled in by the system; otherwise the supplied values will be used, which may be useful when upgrading. *arb1* may be any arbitrary information desired.

`add_status_log` and `del_status_log` may be used instead of or in addition to automatic status logging. It is even possible to log things other than genuine status changes against an item, e.g. when the status of a part changes log this in the arbitrary field of a status log against the CR(s) affecting the part.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

### Example

```
add_status_log('cr', cr_number, cr_class, cr_status, ' ', ' ', "Update Priority to ".cr_Priority)
```

Adds a new status log entry for the current status to log a change to the Priority field of the current CR record.

### *alter\_status\_log*

```
alter_status_log(itemtype, item, olddate, class, status, date, user, arb1)
```

Alters the status log entry identified by *itemtype*, *item* and *olddate* to hold the information specified by *class*, *status*, *date*, *user* and *arb1*, cf. `add_status_log()`.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

#### **Example**

```
alter_status_log('cr', cr_number, '2004/09/06 15:50:16', cr_class, cr_
status,
                "2004/09/07 15:50:16", user, "Modify Priority to ".cr_
Priority)
```

Modifies the status log entry for the current cr with the date of 6th September 2004 at 15:50:16 to the current cr class, the current cr status, for 7th September 2004 at 15:50:16 by the current user with arbitrary field showing a change to the Priority field

### *appendlistvar*

```
appendlistvar(var-name, str)
```

Appends the string *str* to the user-defined variable named *var-name*; the variable is converted to a list (if it is not one already). This is equivalent to appending to the list using `setvar()`, but it is much more efficient.

#### **Example**

```
local(list);
setvar(list, '1');
appendlistvar(list, '2');
listlen(var(list)); #returns 2,
```

Defines an variable called 'list'. Fills the first item in the list with '1' and appends to the end of it a second item, '2'. The length of the list would then evaluate to 2.

### *appendstrvar*

```
appendstrvar(var-name, str)
```

Appends the string *str* to the user-defined variable named *var-name*; the variable is converted to a string (if it is not one already). This is equivalent to `setvar(var-name, var(var-name) . str)` but it is much more efficient.

#### **Example**

```
local(string);
setvar(string, '1');
appendstrvar(string, '2');
echo(var(string)); #outputs 12
```

Defines an string variable called 'string'. sets the string to contain the value '1' and then adds the string '2' to the end of the variable. Finally outputs the finished string.

### *arbitrary\_field\_compulsory*

```
arbitrary_field_compulsory(arbfieldname, class)
```

Returns whether this arbitrary field for items of the class specified must have a value, as defined in the **Field Names** in ACCONFIG. *Class* should be `any` if the arbitrary field applies to any class. *Arbfieldname* must name the real arbitrary field name (and so be quoted), e.g. "pa\_arb1".

This function exists for backward compatibility only and is not recommended for use. Use the appropriate field reference instead.

#### **Example**

```
arbitrary_field_compulsory("cr_arb2", cr_class);
```

Returns whether arbitrary field 2 for the class of the current cr record is compulsory, e.g. False/empty string

### *arbitrary\_field\_datatype*

```
arbitrary_field_datatype(arbfieldname, class)
```

Returns the datatype for items of the class specified, as defined in **Field Names** in ACCONFIG. *Class* should be `any` if the arbitrary field applies to any class. *Arbfieldname* must name the real arbitrary field name (and so be quoted), e.g. "pa\_arb1".

This function exists for backward compatibility only and is not recommended for use. Use the appropriate field reference instead.

#### **Example**

```
arbitrary_field_datatype("cr_arb2", cr_class);
```

Returns the data type e.g. `string`, for arbitrary field 2 for the class of the current cr record.

### *arbitrary\_field\_display*

```
arbitrary_field_display(arbfieldname, class)
```

Returns the display name (as used in ACE) for this arbitrary field for items of the class specified. *Class* should be `any` if the arbitrary field applies to any class. *Arbfieldname* must name the real arbitrary field name (and so be quoted), e.g. "pa\_arb1".

This function exists for backward compatibility only and is not recommended for use. Use the appropriate field reference instead.

#### **Example**

```
arbitrary_field_display("cr_arb2", cr_class);
```

Returns the display name, e.g. "Priority", for arbitrary field 2 for the class of the current cr record.

### *arbitrary\_field\_edit*

```
arbitrary_field_edit(arbfieldname, class)
```

Returns whether this arbitrary field for items of the class specified is editable, as defined in **Field Names** in ACCONFIG. *Class* should be `any` if the arbitrary field applies to any class. *Arbfieldname* must name the real arbitrary field name (and so be quoted), e.g. "pa\_arb1".

This function exists for backward compatibility only and is not recommended for use. Use the appropriate field reference instead.

### Example

```
arbitrary_field_edit("cr_arb2", cr_class);
```

Returns whether arbitrary field 2 for the class of the current cr record, e.g. the empty string/false.

### *arbitrary\_field\_entrytype*

```
arbitrary_field_entrytype(arbfieldname, class)
```

Returns the entrytype for items of the class specified, as defined in **Field Names** in ACCONFIG. *Class* should be `any` if the arbitrary field applies to any class. *Arbfieldname* must name the real arbitrary field name (and so be quoted), e.g. "pa\_arb1".

This function exists for backward compatibility only and is not recommended for use. Use the appropriate field reference instead.

### Example

```
arbitrary_field_entrytype("cr_arb2", cr_class);
```

Returns the entry type, e.g. `combo`, for arbitrary field 2 for the class of the current cr record.

### *arbitrary\_field\_name*

```
arbitrary_field_name(arbfieldname, class)
```

Returns the new field name for this arbitrary field for items of the class specified, as defined in **Field Names** in ACCONFIG. *Class* should be `any` if the arbitrary field applies to any class. *Arbfieldname* must name the real arbitrary field name (and so be quoted), e.g. "pa\_arb1".

### Example

```
arbitrary_field_name("cr_arb2", cr_class);
```

Returns the new field name, e.g. "cr\_Priority", for arbitrary field 2 for the class of the current cr record.

### *arbitrary\_field\_new\_values*

```
arbitrary_field_new_values(arbfieldname, class)
```

Like `arbitrary_field_values()`, but *arbfieldname* must name the mapped, site-defined arbitrary field name (and so be quoted), e.g. "cr\_Priority". Used in `condedit.ac`.

### Example

```
arbitrary_field_new_values("cr_Priority", cr_class);
```

Returns a list of the defined possible values for the arbitrary field "cr\_Priority", used by the class of the current CR record, for example High, Medium and Low as the Priority field in the Intasoft Standard project template.

### *arbitrary\_field\_oldname*

```
arbitrary_field_oldname(arbfieldname, class)
```

Returns the old field name for this arbitrary field for items of the class specified, as defined in **Field Names** in ACCONFIG. *Class* should be any if the arbitrary field applies to any class. *Arbfieldname* must name the mapped, site-defined arbitrary field name (and so be quoted), e.g. "cr\_Priority".

#### Example

```
arbitrary_field_oldname("cr_Priority", cr_class)
```

Returns the original, system defined, field name, e.g. "cr\_arb2", for the arbitrary field "cr\_Priority", defined for the class of the current CR Record

#### *arbitrary\_field\_values*

```
arbitrary_field_values(arbfieldname, class)
```

Returns a list of the defined possible values for this arbitrary field for items of the class specified, as defined in **Field Names** in ACCONFIG. *Class* should be any if the arbitrary field applies to any class. *Arbfieldname* must name the real arbitrary field name (and so be quoted), e.g. "pa\_arb1". Used in `condedit.ac`.

#### Example

```
arbitrary_field_values("cr_arb2", cr_class);
```

Returns a list of the defined possible values for CR arbitrary field 2 defined for the class of the current CR record, for example High, Medium and Low as the Priority field in the Intasoft Standard project template

#### *app\_minimise*

```
app_minimise()
```

Minimises the ACE application. Returns whether it did anything (i.e. false if it was already minimised). This may be useful when ACE is acting as a server to some other application.

#### *app\_restore*

```
app_restore()
```

Restores the ACE application. This may be useful when ACE is acting as a server to some other application.

#### *apply\_read\_perms*

```
apply_read_perms(honour_read_perms)
```

Determines whether AllChange should honour read permissions when evaluating a field reference. The *honour\_read\_perms* parameter should be true or false to denote whether to honour read permissions or ignore them.

This may be useful, for example, in a secure context where the fields' read permissions would normally be ignored to provide the field value, but in some cases this would not be desired.

When AllChange evaluates fields for browser columns, read permissions are normally honoured. This may be changed by use of the `apply_read_perms` function with a parameter of false.

Calling this function outside of a secure context has no effect, i.e. using it outside of secure context can never allow read access where it would otherwise be denied.

***attach\_pseudo\_workspace***

```
attach_pseudo_workspace(toppart, topdir)
```

Creates a temporary *in-memory* workspace, named `_pseudows`, with specified *toppart* and *topdirectory*, hierarchical, and attaches to it. If *toppart* is empty/None, removes workspace and attaches to no workspace. Used when checking in from no workspace

**Example**

```
attach_pseudo_workspace('/', 'C:\tmp')
```

Creates and attaches to a pseudo workspace for the entire part tree related to C:\tmp directory

```
attach_pseudo_workspace('', '')
```

Removes the pseudo workspace and attaches to no workspace (i.e. detaches from any workspace)

***attachment\_dirname***

```
attachment_dirname(str)
```

Takes a string — which could be a CR number or a baseline name — and generates a directory name from it, into which that CR/ baseline's attachments will be stored. Returns the resultant directory name. This is basically the same as the CR number/ baseline name, but with any characters which are not legal in a filename replaced by `_` (underscore).

**Example**

```
attachment_dirname('ACHELP')
```

Returns `ACHELP`, which is the name of the directory which will contain files attached to the baseline named ACHELP.

***baseline\_basename***

```
baseline_basename(baseline-name)
```

 Returns the basename of a baseline name.
**Example**

```
baseline_basename('Baseline1;6.2')
```

Returns 'Baseline1'

***baseline\_template\_text***

```
baseline_template_text(baseline-class)
```

Returns, as a list of lines, the text of the template file, if any, for new baselines of class *baseline-class*.

**Example**

```
baseline_template_text('release')
```

Returns the contents of the template file for baselines of class 'release', this might be for example:

Build Notes::  
Release Notes::

### ***baseline\_text\_filename***

`baseline_text_filename(baseline)`

Returns the filename for the baseline text corresponding to baseline *baseline*. In general this is the same as the baseline name but with any punctuation characters removed. This is the name of the file that will be created by the `getbltext` command and used by the `putbltext` command.

#### **Example**

`baseline_text_filename('My_Baseline;3.3')` will return 'MyBaseline33'

### ***baseline\_text\_section***

`baseline_text_section(baseline-text, section)`

Returns a list of the lines in the specified *section* in *baseline-text*. Note that *baseline-text* is the actual Baseline text (e.g. as produced by the field reference `bl_text` or by reading in a Baseline text template), not a Baseline name.

#### **Example**

`baseline_text_section(bl_text, 'Description')`

Returns the contents of the Description section of the Baseline text for the current Baseline record

### ***baseline\_text\_section\_list***

`baseline_text_section_list(baseline-text)`

Returns a list of all the section names in *baseline-text*. Note that *baseline-text* is the actual Baseline text (e.g. as produced by the field reference `bl_text` or by reading in a Baseline text template), not a Baseline name.

#### **Example**

`baseline_text_section_list(bl_text)`

Returns a list of the text sections names for the Baseline text for the current Baseline record, e.g.

Build Notes  
Release Notes

### ***baseline\_version***

`baseline_version(baseline-name)` Returns the version of a baseline name.

#### **Example**

`baseline_version('Baseline1;6.2')`

Returns '6.2'

***baselined\_version***

```
baselined_version(part, baseline)
```

Returns what version, if any, of *part* appears in *baseline*. The path must be given as a full path name. If *baseline* is a meta-baseline its constituent baselines are followed down recursively. If *baseline* is *any* the function returns a list of the versions of *part* that are contained in all baselines.

**Example**

```
baselined_version('/Parts/icon_test.htm', 'Baseline1;002')
```

Returns the version of a part found in the baseline Baseline1;002, e.g. `'/Parts/icon_test.htm;001'`

```
baselined_version('/Parts/icon_test.htm', any)
```

Returns the list of versions of a part found in all baselines, e.g. `'/Parts/icon_test.htm;001 /Parts/icon_test.htm;002'`

***blinesaffecting\_bline***

```
blinesaffecting_bline(bline)
```

Returns a list of the baselines which affect baseline *bline*, if any.

***blinesaffecting\_file***

```
blinesaffecting_file(file)
```

Returns a list of the baselines which affect file *file*, if any.

***bt\_filename***

```
bt_filename(filename)
```

Returns the name of the Build Thread file corresponding to *filename*.

**Example**

```
bt_filename('c:\actut\product\sw\source\module1.c')
```

Returns `c:\actut\product\sw\source\BT\module1.c`

***calc\_newversion***

```
calc_newversion(version, branch, excludeedits)
```

Returns the new version to succeed the specified version. If a branch is specified, then the function calculates the next available version on a new branch of that name. The *excludeedits* parameter specifies whether to allow the currently edited version to be returned, or whether the version returned should succeed the edited version. If the version passed in names an optimistic edit (i.e. is in the form `<part>;%<-workspace>`) then the function can be used to determine the final 'real' version that will be created if the part is returned.

**Examples**

```
calc_new_version('/part.txt;002', '', false)
```

Returns '/part.txt;003', assuming that version 002 does not already have a successor, or returns '/part.txt;004' if the part is currently checked out for edit.

## *call*

```
call(func-name {, param})
```

Allows a user defined function to be called. *Func-name* is the name of the user defined function, and the other parameters to the `call` function are the parameters to the user defined function. These parameters are received by the called function as user-defined variables named `p1`, `p2`, etc.; they may then be accessed via the normal `var()` function for reading variables, and they behave like `local` variables. Returns the result of the last statement executed in the function (which may be an explicit `return` statement).

## *call\_dde*

```
call_dde(dde-function, str1, str2)
```

Allows certain DDE (Dynamic Data Exchange) functions to be called. ACE (but not ACC) will act as a DDE client to any other DDE server and/ or as a DDE server to any other DDE client. Multiple client and/ or server connections at a time are supported. *Dde-function* must name a DDE function supported by ACE; *str1* and *str2* depend on the function. The supported values for *dde-function* are:

**connect:** connect to a DDE server. *Str1* should name the service, *str2* the topic. The resulting "conversation handle" (as a number) is put in the ACCEL variable `dde_result_str`.

**execute:** send an execution string to the DDE server. *Str1* should be the string to be executed, *str2* the timeout (in milliseconds).

**request:** request an item of information from the DDE server. *Str1* should name the item desired, *str2* the timeout (in milliseconds). ACE supports text format (`CF_TEXT` only), i.e. the requested data must be a string. After a request call the ACCEL variable `dde_result_str` holds the string returned from the server, e.g.:

```
call_dde('request', 'Item', 10000);
echo('Item = '.dde_result_str);
```

An error is raised if the server does not return the information requested.

**poke:** poke an item of information to the DDE server. *Str1* should name the item, *str2* the value (`CF_TEXT` format only).

**query:** query the connection to the DDE server: after issuing this call the ACCEL variable `dde_result_str` will be set to the '*service-name topic-name*' of the conversation, or '' if **AllChange** is not currently acting as a DDE client. This function is useful for determining whether ACE should (re-)start a DDE conversation.

**wait:** wait for the DDE server to send a disconnect message, or for it to send the special execute message `!finished`.

**restore:** restore the DDE (client) conversation handle to that specified by *str1*.

**disconnect:** send the DDE server a disconnect message.

**server:** cause ACE to offer itself as a DDE server. *Str1* should be the service name, *str2* the topic name desired. ACE will then accept a DDE connection request from another program on the specified service and topic.

Suppose you have an editor which can act as a DDE server to edit a file when it receives an appropriate execution message and to send a disconnect message back to the client when the user has finished editing and saved. A suitable sequence of ACCEL statements to get the editor to edit a file and wait for it to finish could be:

```
call_dde('connect', 'EditorService', 'EditorTopic');
call_dde('execute', '[edit '.var(fname).']', 1000);
call_dde('wait', '', '');
```

The editor needs to respond with a DDE acknowledgement message when it receives the execute message if it is prepared to start editing the file and it needs to send a disconnect message when the user has finished editing and saved the file.

Multiple DDE client conversations with other servers are supported. If this is used appropriate client conversation handle must be restored before each `call_dde()` that communicates with a server, e.g.:

```
call_dde('connect', 'Server1', 'Topic1');
setvar(dde1, dde_result_str);
call_dde('connect', 'Server2', 'Topic2');
setvar(dde2, dde_result_str);
call_dde('execute', '[You are Server 2]', 1000);
call_dde('restore', var(dde1), '');
call_dde('execute', '[You are Server 1]', 1000);
```

When acting as a DDE server, ACE will respond to DDE `execute` requests from the client by executing the string passed in as an **AllChange** command. Normally it will try to execute the command and return success or failure to the client according as the command succeeds or fails (i.e. does or does not generate an error); if the command begins with a `&` (ampersand) it will instead return success immediately and run the command asynchronously. If the command begins with a `>` (greater-than) then any error message raised in the course of the command will be placed in the ACCEL variable `error_message` instead of being output to the user; this variable is guaranteed to be set to empty prior to each DDE command executed, and the client may retrieve its value by sending ACE a DDE request message. Only one command can be executed at a time; if **AllChange** is executing any other command it will return *busy*.

When running **AllChange** as a DDE server it is desirable to restrict user interaction with ACE. If the user initiates an operation that updates the database while a similar request happens over DDE then the outcome is unpredictable. ACE may be invoked with a `-minimised` command line option: ACE is minimised and the user is unable to restore it or exit it. The client can send **AllChange** a `&quit` command to cause it to exit.

**AllChange** can act as both a DDE client and server at the same time to the same or different other programs. As a special case, an **AllChange** client which is executing a `call_dde('wait', '', '')` waits either for a disconnect from the server or for a client to send the **AllChange** (acting as a server too) the string `!finished` in an execute message. This makes it possible to set up a two-way communication channel with another program along the following lines (this is a more complex reworking of the earlier example):

```
A/C : call_dde('server', 'AllChange', 'ACTopic')
Other: call_dde('connect', 'AllChange', 'ACTopic')
Other: call_dde('server', 'OtherServer', 'OtherTopic')
A/C : call_dde('connect', 'OtherServer', 'OtherTopic')
A/C : call_dde('execute', '[edit file]', 10000)
Other: <read in file and acknowledge>
A/C : call_dde('wait', '', '')
Other: <let user edit file and save file>
Other: call_dde('execute', '!finished', 10000)
A/C : <continue work, presumably saving file in database>
```

**AllChange** will respond to DDE request requests from the client by evaluating the string passed in as an ACCEL expression. The result of the evaluation is returned as a text string (`CF_TEXT`) to the client. This makes it possible for clients to request a great variety of information from **AllChange**.

NOTE: The same note as under `call_dll` applies to using `call_dde`.

### Example

```
call_dde(connect, 'Excel', 'system');
```

Connects to an Excel DDE server with 'system' as the topic.

### *call\_dll*

```
call_dll(str1, str2, str3, str4, str5, str6)
```

Allows an arbitrary Windows DLL (Dynamic Link Library) call from either ACE or ACC. ACE/ACC calls a function by the name of `ACCa11D11` in a DLL called `ACINTDLL.DLL` with the supplied 6 arbitrary string arguments. This function may then perform whatever action is desired. The result is the number returned by `ACCa11D11`.

By convention, any function in any (other) DLL may be called by specifying the DLL name as *str1* and the function name as *str2*. A sample `ACINTDLL.DLL` is supplied with the distribution which does just this to call the Microsoft Mail DLL to implement a MAPI mail command for **AllChange**; `call_dll` should be used to invoke it as follows:

```
call_dll('ACMAPIEX.DLL', 'MAPIExSendMail', 'Fred', 'Subject', 'Some
        text', '')
```

The C source file `ACINTDLL.C` is also supplied and may be tailored to produce a DLL which implements any desired functionality including calling other DLLs. The comments explain how `ACINTDLL.C` may be edited to enhance its functionality; they also explain how to achieve this by creating a site-maintained DLL instead, which avoids having to change `ACINTDLL.C` or `ACINTDLL.DLL` in order to ease upgrading.

The `ACCa11D11` function called by ACE in this source file is passed a pointer to a function in the **AllChange** executable program. This function accepts a text string passed to it as an ACCEL expression and returns the result of evaluating the expression. This makes it possible for DLLs to request a great variety of information from **AllChange**. The function is also passed a pointer which can be used to set the ACCEL variable `dll_result_str` to any desired string. See the source file for further information.

As a special case, if *str2* is `LockDLLInMemory` then the DLL named by *str1* is locked into memory; it will be unlocked on exit from ACE. This facility has been introduced to speed up calls to DLLs which take a long time to initialise themselves: by putting this call in the `ac.ini` file the DLL will be locked in memory all the time ACE is running. DLLs locked in memory are automatically freed on exit from ACE.

NOTE: This function provides a very powerful general purpose interface to DLLs. However, the Administrator is responsible for ensuring that any DLLs called do not interfere with the functioning of **AllChange**. Intasoft support cannot help you with writing or interfacing to DLLs. In addition, Intasoft support cannot help you with **AllChange** problems which, in their opinion, might arise from using DLLs.

### *called\_from*

```
called_from(command)
```

Returns whether the current command was invoked from the *command* specified. The *command* may be either the name of an **AllChange** underlying command or the name of a function. It will return true if the current command was invoked directly or indirectly from the command specified and false otherwise.

This function is useful in entry conditions for command definitions to enforce that a command is invoked only from another and not directly by the user. For example, it may be used to prevent check-outs (issue command) except via the **status** command, thus enforcing a life-cycle driven approach to using the system.

### Example

```
called_from('AssignPartsaffectedOnReturn')
```

Would return true if the current command was invoked directly or indirectly from 'Assign-PartsaffectedOnReturn'

## *call\_ole*

`call_ole(ole-function, str1, str2, str3)`

Allows certain OLE Automation functions to be called, with ACE/ACC acting as an OLE Automation client and/ or acting as an OLE Automation server. Acting as an Automation client allows **AllChange** to drive and/ or extract information from other programs which act as OLE Automation servers, such as Microsoft Project ; similarly, acting as an Automation server allows other programs to drive and/ or extract information from **AllChange**. Multiple connections at a time are supported.

OLE Automation is in many respects similar to DDE. It may be used in place of DDE where a program can act as an OLE Automation client/server but not as a DDE client/server, or if the OLE Automation interface is preferable to program.

**AllChange** calls a function by the name of `OLECommand` in the supplied DLL `ACINTOLE.DLL` with the *ole-function* and 3 arbitrary string arguments. The ACCEL variable `ole_result_str` is set to the result, if any, returned from the last `call_ole()`. The recognised *ole-commands*, meaning of the subsequent arguments (unused arguments should be passed as ' '), returned `ole_result_str`, and equivalent Visual Basic function are as follows:

### **CreateObject, progID**

Creates a new OLE Automation object. *Str1* is the desired object's progID (in the form *appName.objecttype*). The application which creates the object is started if not running. The resulting object is returned in `ole_result_str`; this should be saved and used for accessing the object's properties and methods. This is equivalent to VB's `Set Obj = CreateObject()`.

### **GetObject, filename, progID**

Accesses an OLE Automation object from a file, or finds a currently active object. *Str1* is the filename of the desired object; *str2* is the desired object's progID. Either *str1* or *str2* may be empty ( ' '), but not both. If *str1* is not ' ' the application associated with the specified file is started; then the object specified by *str2* (or the default object if *str2* is ' ') is activated. If *str1* is ' ' a (random) currently active object with progID specified by *str2* is found. The resulting object is returned in `ole_result_str`; this should be saved and used for accessing the object's properties and methods. This is equivalent to VB's `Set Obj = GetObject()`.

### **ReleaseObject, object**

Releases an object obtained from a previous `CreateObject`, `GetObject` or `GetSubObject`. Objects should be released when no longer needed. Equivalent to setting an object variable to something else (including `Nothing`) or letting it go out of scope in VB. Note that objects should be explicitly released in ACCEL.

### **ShutDown**

Terminates all OLE Automation activity. Automatically called before exiting **AllChange**. Can be called explicitly to ensure clean state. New activity can be restarted with `CreateObject` or `GetObject`.

### **GetSubObject, object, method**

Calls the method named in *str2* of the object specified in *str1*. This method must return a new object. The resulting object is returned in `ole_result_str`; this should be saved and used for accessing the object's properties and methods. This is equivalent to VB's `Set NewObj = object.method`.

### **Method or Call, object, method**

Calls the method named in *str2* of the object specified in *str1*. *Ole-command* may be either `Method` or `Call` as preferred. If the method returns anything it will be converted to a string and stored in `ole_result_str`. This is equivalent to VB's `object.method` or `Var = object.method`.

### **Procedure, object, method**

Like `Method` or `Call`, but does not let the method return any result. This is equivalent to VB's

`object.method`. May be required by some OLE Automation servers which insist that its methods which do not return anything be called this way.

**Get, *object*, *property***

Retrieves the property named in *str2* of the object specified in *str1*. The property will be converted to a string and stored in `ole_result_str`. This is equivalent to VB's `Var = object.property`.

**Put or Set, *object*, *property*, *value***

Sets the property named in *str2* of the object specified in *str1* to the value specified in *str3*. *Ole-command* may be either `Put` or `Set` as preferred. This is equivalent to VB's `object.property = value`.

**Server, *automation*, *progID***

Instructs ACE to start acting as an OLE Automation server.

A simple illustration of the client facilities is:

```
call_ole('CreateObject', 'Excel.Application', '', '');
setvar(xlApp, ole_result_str);
call_ole('Put', var(xlApp), 'Visible', true);
call_ole('Method', var(xlApp), 'Quit', '');
```

When acting as an OLE Automation client, if the server does not complete an operation within a short period the user will be presented with a **Server not responding** dialog. Although the dialog allows a retry, it can be a bit disconcerting for the user. The default timeout is 30 seconds, which should be adequate for most purposes. However, for example, the time taken for Word to stamp a very large document with **All-Change** fields during check in could exceed this threshold. The timeout may be increased, if necessary, by creating a registry entry in the **AceSettings** section (use `ACPRJSET` to accomplish this). It should be a DWORD value named **OleTimeout** whose value is the timeout in milliseconds (e.g. 60000 for one minute).

When *ole-command* is `Server` ACE/ACC starts acting as an OLE Automation server. *Automation* should be set to 1 (or true) if ACE/ACC is being started in response to a client's "CreateObject" call, in which case `EAUTOMATION=1` will have been passed on the command-line to ACE/ACC, otherwise it should be '' (or 0 or false). If *Automation* is set to **Exclusive** (i.e. `-EAUTOMATION=Exclusive`) then the ACE/ACC will serve *only* the invoking Automation client (i.e. it cannot be connected to by another application using "GetObject").

*ProgID* may be set to the progID ACE/ACC should register; if this is '' then the default, `AllChange.Application`, is used. During workstation installation 10 progIDs, `AllChange.Application` plus `AllChange.Application_[1-9]`, are created for ACE, plus `AllChange.ACC.Application` for ACC, in the registry. Normally a container should connect to the default `AllChange.Application` but, in view of the fact that ACE should act as an OLE Automation server to no more than one container at a time, any of the 9 other alternatives may be used to ensure a given application communicates with a unique instance of ACE; this is similar to ACE's ability to offer different DDE server topics. If ACC is invoked with the command-line argument `-EACOLESERVER=1` it will act *only* as an OLE Automation server, i.e. it will not respond to commands from the console prompt; it will also only use HKLM, not HKCU, in the registry.

As an Automation server ACE/ACC offers just two methods:

- `Execute` (*AllChange-command*)
- `Request` (*ACCEL-expression*)

These are just like when ace acts as a DDE server and cause ace to execute a command or evaluate an ACCEL expression respectively, e.g. (in VB):

```
Dim AllChangeApp As Object
Set AllChangeApp = CreateObject( , "AllChange.Application")
AllChangeApp.Execute("attachws myws")
Print "Current workspace is: " & AllChangeApp.Request("workspace")
AllChangeApp.Execute("&quit")
```

```
Set AllChangeApp = Nothing
```

In view of the simplicity of the interface we have not bothered to supply a Type Library.

**AllChange** workstation installation creates the required registry entries, and uninstallation (when removal of the Intasoft program group and registry settings is selected) removes them. The supplied `ac.ini` file causes ACE to offer itself automatically as an OLE Automation server using the default progID.

More advanced examples may be found in the supplied `olefunc.ac` file, which is just a sample file. This file also gives a detailed explanation of how to use all the facilities provided in **AllChange** OLE Automation. The C source file `ACINTOLE.C` is also supplied and could be tailored to alter/ enhance the functionality provided through `call_ole()`.

NOTE: The same note as under `call_dll` applies to using `call_ole`.

### Examples

```
call_ole('ReleaseObject', var(obj), '', '')
```

Releases an OLE object obtained from a previous `CreateObject`, `GetObject` or `GetSubObject`.

### *call\_stack*

```
call_stack()
```

Returns the ACCEL call stack, which comprises the names of all inbuilt command and ACCEL function calls which lead to the current context (in reverse order).

### *can\_pass\_vote\_next*

```
can_pass_vote_next(database, item, status, error)
```

Determines whether a serial vote may be passed to the next set of voters at this time (by a user with the appropriate permissions). If 'error' is true, then the function throws an error with a message to describe why the pass to next is not currently allowed, otherwise the function simply returns true to indicate that passing to next is allowed or false to indicate that it is not.

### Example

```
can_pass_vote_next('cr', 'RFC00123', 'AssessEval', false)
```

Returns whether the vote on RFC00123 in the AssessEval status may be passed to the next voters if this is a serial vote.

### *check\_arb\_field*

```
check_arb_field(arbfieldname, class, adding)
```

Checks to see whether the current command — which should be one trying to add or alter a record in a database which has arbitrary fields — complies with any requirements for an arbitrary field's value, as specified in **Field Names** in `ACCONFIG`, and raises an error if not. *Arbfieldname* must name the real arbitrary field name (and so be quoted), e.g. "pa\_arb1"; *class* should name the class to which the arbitrary field applies, or any if it applies to any class; *adding* is a truth indicating whether the command is adding a new item into the database rather than altering an existing item.

This function ensures that a valid value is supplied for an arbitrary field which only accepts one of a list of predetermined possible values; if *adding* is true it also ensures that any arbitrary fields which are compulsory have indeed been supplied with a value. The entry condition of commands which add or alter records call this function on every possible arbitrary field in turn

**Example**

```
check_arb_field("pa_arb1", any, true)
```

Validates the arbitrary field value supplied for arb1 when adding a part of any class.

***check\_monitor***

```
check_monitor(item, event)
```

Checks to see whether any monitors have been placed on *item* for *event* and, if so, triggers them.

**Example**

```
check_monitor(pa_part, 'part')
```

Checks if there are any monitors for event 'part' for the current part, and if so triggers them

***check\_rolemap***

```
check_rolemap(action, item, part-name, item-class, owner-list)
```

Checks whether the user has the required permission to execute the specified *action*, within the area of the parts tree rooted at *part-name* for *item* which is of class *item-class*.

If the *action* is the name of an ACCEL function for Function type entries, then an *owner-list* may also be specified as a list of user names. If the corresponding Command Access (rolemap) entry includes the role (**Owner**) then the user must be in the *owner-list* or the list must be empty in order to have the required permission to invoke the function.

The *item* parameter is only used when checking dynamic roles (e.g. bl\_user, cr\_assignee, etc), and only for built-in commands and not functions. If no item is specified, then the current record of the appropriate type is used. Specifying an item will cause the item to be retrieved from the database, and so, for speed, the current record should be used, if there is one, in preference to specifying the item.

The *item* should be an appropriate item for the command being tested, so for alterbaseline, the item should specify a baseline. For issue/check-out commands using iu\_user, the item should be the workspace name and the full part path, separated by a tab character, for instance "srcws^t/product/sw/module.c;002". For monitor commands using mo\_user, the item should specify the four pieces of information, again separated by tabs, needed to identify the monitor record to check, i.e. "item^tevent^tuser^tarb1", e.g. "/products/module1.c^tpart^tfred^treturnedit". The monitor 'item' must be specified, but the other pieces of information may be optional by using 'any', or omitting them from the end of the string, so "/product/module1.c^tany^tfred" will use the first monitor entry found with the specified item and user.

The required permissions are defined in the appropriate entry in the `rolemap.ac` configuration file (accessed from **Access | Command Access** in `acconfig`).

**Example**

```
check_rolemap('RenameCR', var(old), cr_toppart(var(old)), cr_
class(var(old)), call(CROwners, var(old)));
```

Returns whether the current user has permission to rename a CR for the original CR for the CRs topart and the class of the CR. If the permissions specify (Owner) then the user must also be a member of the list returned from the function CROwners.

***check\_rolemap\_user***

```
check_rolemap_user(action, item, part, class, owner-list, options-list, username)
```

Returns whether *username* could have permission to perform *action* on *item* within the area of the parts tree rooted at *part* for the *class* specified with the command options (*option-list*) as specified. This uses the Command Access (rolemap.ac) settings to determine the permission.

Each option in *option-list* may be prefixed by ! to mean not that option. If *class* is '**any**' then this means any class. For commands which are functions, if (**Owner**) is specified in the Command Access for the function then the user must be in the *owner-list* specified.

This is similar to [check\\_rolemap](#) but allows a specified user rather than the current user.

If *username* is empty then the current user is used.

If *item* is empty then the function returns true if the *username* could have permission to perform the command on some items.

If *part* is empty then the function returns true if the username has the required role for some part of the tree

### Example

```
check_rolemap_user('issue', '/SWProduct', 'ControlledSource', '', 'edit',
'Sarah Smith')
```

Returns true if Sarah Smith has permission to issue for edit parts in /SWProduct of class ControlledSource.

### *check\_valid\_branch\_name*

```
check_valid_branch_name(branch)
```

Checks that the specified branch name is valid. Errors if the name is invalid in any way. This function can be used to validate user-entered branch names. The function has no return value.

### Examples

```
check_valid_branch_name("NewBranch")
```

does not error, as it is a valid branch name.

```
check_valid_branch_name("New Branch")
```

produces error "Invalid branch name: name may only contain letters, digits and \_ - (New Branch)", as branch names cannot contain spaces.

```
check_valid_branch_name("1stBranch")
```

errors as "Invalid branch name: first character must be letter or \_ - (1stBranch)".

```
check_valid_branch_name("Exceeding_Lengthy_Branch_Name_For_An_Example")
```

errors with "Invalid branch name: length exceeds 40 characters - (Exceeding\_Lengthy\_Branch\_Name\_For\_An\_Example)"

```
check_valid_branch_name("")
```

produces error "Invalid branch name: name is empty - ()"

### *check\_view\_edits*

```
check_view_edits()
```

Checks to see if any edits have been made in any of the viewers in ACE. If any have been made offers to continue or cancel. If no edits have been made, or if the continue dialog is confirmed, then returns true; if Cancel is selected then returns false.

### *check\_open\_votes*

`check_open_votes()`

This function may be used to cause all open votes to be checked against their deadlines, and other criteria, to see if the vote should be closed. If the vote should be closed, then this function will close it, and the relevant decision/blocked emails are sent.

This function may be called at any time, and is useful, for instance, to call on a regular basis by a scheduled job.

### *child\_list*

`child_list(part)`

Returns a list of the children of the part *part*. The part should be given as a full path name. The child list does not include versions. If a part which is not of type *subsystemis* specified, the list will be empty.

#### **Example**

```
child_list('/Parts/Subdir')
```

Might return `part1 part2 part3`

### *class\_default*

`class_default(type)`

Returns the default class (if any) for items of type *type*.

#### **Example**

```
class_default('component')
```

Might return `'Source'` if that has been set as the default class for components.

### *classes\_of\_type*

`classes_of_type(type-list)`

Returns a list of all classes which pertain to items whose type is among the types in *type-list*. *Type-list* may contain any of the type-names used in the type field of the classes database.

#### **Example**

```
classes_of_type('component subsystem')
```

Would return the names of all classes defined for `components` or `subsystems`.

### *clear\_output*

`clear_output()`

Clears the output window. Note any output currently sitting in a redirection file will still be shown after a call to `clear_output`.

### *clear\_secure\_context*

`clear_secure_context()`

Clears the `secure_context` variable for the current operation. This can be used to ensure that actions invoked directly by a user can be prevented, while allowing the same action to be performed by ACCEL code executed within a secure context. This only has an effect if the called code is designed to recognise whether it is being run in a secure context.

### Example

```
clear_secure_context();
interpret('renamebaseline -to '.quote_arg(var(to)).' -from '.quote_arg(var(from)));
```

Clears the secure context prior to invoking the `renamebaseline` command so that the `renamebaseline` command will be executed as if directly invoked by the user.

### *cmpfiles*

```
cmpfiles(file1, file2)
```

Returns whether the contents of `file1` and `file2` are the same. (Note that this is *much* faster than running the **Diffs** tool and testing the exit code.)

### Example

```
cmpfiles('c:\temp\test1.ac', 'c:\temp\test2.ac')
```

Would return true if both files were the same.

### *command*

```
command(os-command)
```

Issues the operating system command given as the string *os-command*. The function will return the success or failure of the operating system command (i.e. whether the exit code of the command was equal to the operating system's indication of success, usually 0). It also sets the variable `cmd_result` to the exit code returned by the operating system.

Certain punctuation characters at the start of *command* have special significance and are stripped off and acted upon before issuing the command, as follows.

If *os-command* has a leading @ (at) character, then the command string will be echoed prior to issuing the command.

If *os-command* has a leading + (plus) character, then if the command fails a fatal error is generated. In the normal case, the failure of the *os-command* does not cause an error, and processing continues to the next statement.

If *os-command* has a leading ! (exclamation) character the command is executed as an "interactive" command. This has no effect when running ACC(all commands are effectively interactive). When running ACE, commands are normally run with output redirected to a file so that it may be displayed in the Output Window when the command terminates. Commands which are interactive (e.g. the invocation of an editor, and all normal Windows programs) *must* be written with a leading !; the interpreter runs such commands without their output being redirected. The ! has the same effect as *not* selecting the paged button in the ACE Command menu (with the corresponding "hang" if it is not used when needed).

Two further leading punctuation characters with special significance may follow a leading !. If *os-command* starts with !& (ampersand) the "interactive" command is executed but **AllChange** does not wait for it to finish; this allows programs to be run from, say, the `ac.ini` startup file. If *os-command* starts with !% (percent) the "interactive" command is executed using an alternative wait-for-exit method. This method always returns 0 but works correctly with programs that are not 16-bit Windows tasks (e.g. Windows 95

notepad) without giving "Can't find task handle" errors. Note that the & and % must come after any other special characters.

### Example

```
command('!&'.var(visdiffsprog).' '.var(file1).' '.var(file2));
```

Runs the program specified in the variable `visdiffsprog` (e.g. the supplied VisDiffs tool) to compare `file1` and `file2`. This is expected to be an interactive command and **AllChange** will not wait for it to terminate.

### *control*

```
control(args, vcfile)
```

Calls the VC **Control** tool with arguments *args* on the VC file *vcfile*.

For information on valid arguments see the **AllChange VC Tools** manual.

This function *must* be used instead of `command`.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

### *conv\_path*

```
conv_path(path)
```

Converts an operating system path to a similar **AllChange** part. This changes all \ characters to / characters. The result is the converted path.

### Example

```
conv_path('c:\temp\TEST.ac')
```

Converts 'c:\temp\TEST.ac' to '/temp/TEST.ac'

### *conv\_year4*

```
conv_year4(date-string)
```

Converts a date string in the format *yy/mm/dd* or *yy/mm/dd hh:mm:ss* so that it starts with *yyyy/mm/dd*, i.e. to have a 4 digit year, returning the resulting string. If the date string already starts with *yyyy/mm/dd* it is returned unchanged. May be useful for converting old standard format **AllChange** date strings to new standard format.

### Example

```
conv_year4('03/04/06')
```

Returns '2003/04/06'

### *copy\_clipboard*

```
copy_clipboard()
```

Copies from the current edit control to the clipboard.

### *copyfile*

```
copyfile(fromfile, tofile)
```

Copies the file *fromfile* to the file *tofile*. The file's permissions and timestamp are retained. A fatal error is raised if the copy fails.

### ***copyserverfile***

`copyserverfile(fromfile, tofile)`

If running client/server gets the server to copy the file *fromfile* to the file *tofile*; *fromfile* and *tofile* should specify paths from the server's point of view. If not running client/server acts the same as `copyfile()`.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

### ***copyvcfile***

`copyvcfile(fromvcfile, tovcfile)`

Copies the VC file *fromfile* to *tovcfile*.

This function *must* be used instead of `copyfile`.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

### ***createdir***

`createdir(dirname, parent)`

Creates the directory *dirname*. If *parent* is true all non-existing parent directories of *dirname* are created first, otherwise only *dirname* is (attempted to be) created.

### ***createserverdir***

`createserverdir(dirname, parent)`

If running client/server gets the server to create directory *dirname*, and all non-existent parent directories if *parent* is true; *dirname* should specify a path from the server's point of view. If not running client/server acts the same as `createdir()`.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

### ***createvcdir***

`createvcdir(vcdir)`

Creates the VC directory *vcdir*. The VC subdirectory will also be created as necessary.

This function *must* be used instead of `createdir`.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

### ***cr\_part\_versionssolved***

`cr_part_versionssolved(cr, part)`

Returns a list of those versions of *part* (which should be a component) which are among the versionssolved of *cr*. This can be much more efficient than searching through `cr_versionssolved`.

### **Example**

```
cr_part_versionssolved('CR00002', '/Parts/part2.txt')
```

Might return `'/Parts/part2.txt;001'`, if this is in the versionssolved list for `'CR00002'`

### ***crsaffecting***

`crsaffecting(part, statuses)`

Returns a list of the CR numbers whose status is among *statuses* which affect *part*, if any. *Part* may refer to a subsystem, component or version. If *part* specifies a component name terminated by ; then the list of CRs will include all those which affect *any* version of *part*. *Statuses* is a list of the required statuses for the CR; if it is empty any status is acceptable.

**Example**

```
crsaffecting('/Parts/part2.txt', '')
```

Returns a list of the CRs (in any status) which have '/Parts/part2.txt' as a part affected.

***crsaffecting\_bline***

```
crsaffecting_bline(bline, statuses)
```

Returns a list of the CR numbers whose status is among *statuses* which affect baseline *bline*, if any. *Statuses* is a list of the required statuses for the CR; if it is empty any status is acceptable.

**Example**

```
crsaffecting_bline('MetaBaseline1', '')
```

Returns a list of the CRs (in any status) which have "MetaBaseline1" as a baseline affected.

***crsaffecting\_cr***

```
crsaffecting_cr(cr, statuses)
```

Returns a list of the CR numbers whose status is among *statuses* which affect CR number *cr*, if any. *Statuses* is a list of the required statuses for the CR; if it is empty any status is acceptable.

**Example**

```
crsaffecting_cr('CR00006', '')
```

Returns a list of the CRs (in any status) which have "CR00006" as a CR affected.

***crsaffecting\_file***

```
crsaffecting_file(file, statuses)
```

Returns a list of the CR numbers whose status is among *statuses* which affect file *file* (i.e. where *file* is an attachment), if any. *Statuses* is a list of the required statuses for the CR; if it is empty any status is acceptable.

**Example**

```
crsaffecting_cr('screendump.gif', '')
```

Returns a list of the CRs (in any status) which have 'screendump.gif' as an attachment.

***crssolved***

```
crssolved(version, statuses)
```

Returns a list of the CR numbers whose status is among *statuses* which were solved by *version*, if any. *Version* should normally refer to a version; if it refers to a component instead all CRs solved by *any* version of the component are returned. *Statuses* is a list of the required statuses for the CR; if it is empty any status is acceptable.

#### Example

```
crssolved('/Parts/part2.txt;002', '')
```

Returns a list of the CRs (in any status) which have version 002 of /Parts/part2.txt as a version solved.

#### *crssolved\_by\_bline*

```
crssolved_by_bline(bline, statuses)
```

Returns a list of the CR numbers whose status is among *statuses* which were solved by baseline *bline*, if any. *Statuses* is a list of the required statuses for the CR; if it is empty any status is acceptable.

#### Example

```
crssolved_by_bline('MetaBaseline1', '')
```

Returns a list of the CRs (in any status) which have 'MetaBaseline1' as a baseline solved.

#### *cr\_template\_text*

```
cr_template_text(cr-class)
```

Returns, as a list of lines, the text of the template file, if any, for new CRs of class *cr-class*.

#### Example

```
cr_template_text('cr')
```

Returns the contents of the template file for CRs of class 'cr', this might be for example:

```
Description/Reason for Change::
Operational Impact::
Identify any Risk Areas::
Cost Implications::
Workaround (if any)::
Implementation Action::
```

#### *cr\_text\_filename*

```
cr_text_filename(crnumber)
```

Returns the filename for the CR text corresponding to CR number *crnumber*. In general this is the same as the CR number but with any punctuation characters removed. This is the name of the file that will be created by the `getcrtext` command and used by the `putcrtext` command.

### Example

```
cr_text_filename('SCR00005') will return 'SCR00005'
```

```
cr_text_filename('My/CR-1234') will return 'MyCR1234'
```

### *cr\_text\_section*

```
cr_text_section(cr-text, section)
```

Returns a list of the lines in the specified *section* in *cr-text*. Note that *cr-text* is the actual CR text (e.g. as produced by the field reference `cr_text` or by reading in a CR text template), not a CR number.

### Example

```
cr_text_section(cr_text, 'Description')
```

Returns the contents of the Description section of the CR text for the current CR record

### *cr\_text\_section\_list*

```
cr_text_section_list(cr-text)
```

Returns a list of all the section names in *cr-text*. Note that *cr-text* is the actual CR text (e.g. as produced by the field reference `cr_text` or by reading in a CR text template), not a CR number.

### Example

```
cr_text_section_list(cr_text)
```

Returns a list of the text sections names for the CR text for the current CR record, e.g.

```
Description  
Workaround  
Action  
Review Comments
```

### *current\_vote\_users*

```
current_vote_users(database, item, status, hidevotedusers)
```

Returns the voters currently valid to vote as a list of usernames. For a serial vote, this function returns just the voters valid to vote in the current voter set. If *hidevotedusers* is true, only users yet to vote are included, otherwise all valid voters are included (this makes the function much the same as [valid\\_voters](#), except when used for serial votes).

### Example

```
current_vote_users('cr', 'RFC00010', 'AssessEval', true)
```

Returns the voters currently valid to vote on RFC00010 in the AssessEval status who have not yet voted.

### *cut\_clipboard*

```
cut_clipboard()
```

Cuts from the current edit control to the clipboard.

### *date\_compare*

```
date_compare(date1, op, date2)
```

Returns the result of comparing *date1* against *date2* with operator *op*. *Date1* and *date2* must be in the format accepted by `date_to_days()`. *Op* may be any of the following strings:

- Before or <
- On or Before or <=
- On or ==
- On or After or >=
- After or >

#### **Example**

```
date_compare(pa_date, 'After', '2004/04/03')
```

Returns whether the date of the version for the current part/version record is after 3rd April 2004

### *date\_to\_days*

```
date_to_days(date)
```

Returns a number of days elapsed since 1970/01/01 corresponding to *date*.

The date should be specified in the format **YYYY/MM/DD hh:mm:ss** (this is the *standard Intasoft date format*).

#### **Example**

This number may be compared against a similar number to compare one date against another, e.g.:

```
if date_to_days(pa_date) == date_to_days(iu_date) then ...
if date_to_days(pa_date) > date_to_days('1998/01/01') then ...
```

It is also permissible to add to or subtract from this number of days to obtain a different date. For example, to look for versions created more than 90 days (3 months) ago:

```
date_to_days(pa_date) < date_to_days(sys_date(all)) - 90
```

### *date\_to\_seconds*

```
date_to_seconds(date):
```

Returns the number of seconds elapsed since 1970/01/01 00:00:00 UTC (GMT) corresponding to *date*. *Date* should be specified in standard Intasoft format (YYYY/MM/DD hh:mm:ss).

#### **Example**

```
date_to_seconds('1970/01/01 00::00::20')
```

Returns 20, the number of seconds elapsed since '1970/01/01'

### *days\_to\_date*

```
days_to_date(date)
```

Returns the date corresponding to a number of days elapsed since 1970/01/01. The date will be in standard Intasoft format (YYYY/MM/DD). The number of days should have been produced from `date_to_days()`.

### Example

```
days_to_date(10)
```

Returns '1970/01/10', showing date 10 days after 1970/01/01.

### *db\_tran\_flush*

```
db_tran_flush()
```

Flushes any transaction(s) held in memory to the database. Such transactions can still be backed out, but they are visible to the outside world. May be used when an action requires the database to be up-to-date.

### *db\_tran\_flush\_no\_backout*

```
db_tran_flush_no_backout()
```

Flushes any transaction(s) held in memory to the database and removes from the list of transactions which may be backed out (i.e. these transactions will not be backed out on an error). May be used when an action requires transactions not to be undone.

### *default\_index\_value*

```
default_index_value(database, index)
```

Returns the default value for this *database/index* pair. *Database* must identify a real database, like one passed to `num_records()`; *index* must identify a display-type index name, like one returned by [display\\_index\\_names\(\)](#). Suitable for offering in a report dialog.

### *deactivate\_app*

```
deactivate_app()
```

Deactivates the ACE application, i.e. allows another application to be brought to the foreground. This may be useful when ACE is acting as a server to some other application and wants to allow that application to interact with the user.

### *defined*

```
defined(var-name)
```

Returns whether the user-defined variable *var-name* is currently defined.

### Example

```
defined(Global_CheckInCount)
```

Would return true if `Global_CheckInCount` had been defined.

### *def\_version*

```
def_version(part)
```

Returns the default version of the part *part*. The part should be given as a full path name.

**Example**

```
def_version('/Parts/part1.txt')
```

Would return `/Parts/part1.txt;004` if that were the default version

***deldir***

```
deldir(dirname, contents)
```

Deletes directory *dirname*. If *contents* is true all files and subdirectories of *dirname* are deleted as well; if *contents* is false only *dirname* is deleted (and so must be empty). Use with care.

***delfile***

```
delfile(filename)
```

Deletes the file *filename*. This function deletes a file even if it is read-only. A warning message is issued if the file exists but could not be deleted.

***delserverdir***

```
delserverdir(dirname, contents)
```

If running client/server gets the server to delete directory *dirname*, and all its contents if *contents* is true; *dirname* should specify a path from the server's point of view. If not running client/ server acts the same as `deldir()`.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

***delserverfile***

```
delserverfile(filename)
```

If running client/server gets the server to delete file *filename*; *filename* should specify a path from the server's point of view. If not running client/server acts the same as `delfile()`.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

***del\_status\_log***

```
del_status_log(itemtype, item, condition)
```

Deletes logs of *item*, of type *itemtype*, matching ACCEL*condition* from the status log database. If *item* is empty all items of type *itemtype* are candidates for deletion. If *condition* is not empty only those selected items satisfying *condition* are deleted; the current database record is the item in question.

**Example**

```
del_status_log(part, '/subsys/comp1;004', '')
    # delete all status logs of part /subsys/comp1;004

del_status_log(cr, '', 'date_to_days(cr_status_log_date) <
    date_to_days("1998/01/01")')
    # delete all CR status logs older than 1/1/1998
```

Use with care. `add_status_log` and `del_status_log` may be used instead of or in addition to automatic status logging.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

### *delvmdir*

`delvmdir(vmdir)`

Deletes the VC directory *vmdir*. The VC subdirectory will also be deleted as necessary.

This function *must* be used instead of `delmdir`.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

### *delvcfile*

`delvcfile(vcfilename)`

Deletes the VC file *filename*.

This function *must* be used instead of `delcfile`.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

### *dir\_exists*

`dir_exists(dirname)`

Returns whether the directory *dirname* exists.

#### **Example**

```
dir_exists('c:\Temp')
```

Would return true if 'c:\Temp' existed

### *dir\_size*

`dir_size(directory, recursive)`

Returns the size of files contained in the specified directory. If recursive is true then sizes of sub-directories are also included recursively.

#### **Example**

```
dir_size("C:\tmp", false)
```

Might return 7244 as the size of the files in the directory `C:\tmp` but excluding any subdirectories.

### *dirname*

`dirname(path)`

Returns the (parent) directory of operating system path *path*.

#### **Example**

```
dirname('c:\Temp')
```

Returns 'c:\'

### *display\_index\_names*

`display_index_names(database, listall)`

Returns a list of the names of all the fields which may be used to index the database *database*. *Database* must identify a real database, like one passed to [num\\_records\(\)](#). *listall* is a truth value indicating whether to list all available indexes, or only those compatible with a `-latest` parameter: This is useful with baseline versions. Each name in the returned list is in the display format suitable for offering to the user in a list box. The name must be converted to a real index name via [display\\_index\\_to\\_real\\_index\(\)](#) before using as a command argument.

### Example

```
display_index_names('cr', true)
```

Might return the names for all fields defined for the CR database/table including arbitrary fields plus the text field.

### *display\_index\_to\_real\_index*

```
display_index_to_real_index(database, index)
```

Converts from a display-type index name to a real index name suitable for passing to an **AllChange** command. *Database* must identify a real database, like one passed to [num\\_records\(\)](#); *index* must identify a display-type index name, like one returned by [display\\_index\\_names\(\)](#). Returns the real index name.

### Example

```
display_index_to_real_index('cr', 'Status')
```

Returns `status`

### *display\_index\_values*

```
display_index_values(database, index)
```

Returns a list of possible values to offer the user for this *database/index* pair. *Database* must identify a real database, like one passed to [num\\_records\(\)](#); *index* must identify a display-type index name, like one returned by [display\\_index\\_names\(\)](#).

### Example

```
display_index_values('cr', 'Status')
```

Might return `Accepted Closed Complete ..etc.`

### *do\_dialog*

```
do_dialog(dllname, dlgnum, dlgproc, helpfile, helpstr) :
```

Invokes a modal user-defined dialog, and waits for the user to interact with it and exit.

- *dllname* specifies the name of the DLL file containing the dialog (default `ACUIDLG.DLL`);
- *dlgnum* specifies the resource ID of the dialog in the DLL to be displayed;
- *dlgproc* specifies the name of the dialog box procedure in the DLL for the dialog (default `ACUIDlgProc`);
- *helpfile* and *helpstr* provide a link to the Help system. For a Visual ACCEL dialog which has a **Help** button, if *helpfile* is the empty string help will be the dialog whose ID equals *dlgnum* + 1000, as described in [Visual ACCEL](#). Otherwise, *helpfile* should give the name of the Help file with which

HTML Help is to be invoked: if *helpstr* is empty it will jump to the contents of the help file, otherwise it will jump to the topic with that context string.

### Example

```
do_dialog('', 1001, '', join_paths(acexedir, 'acmanual.chm'), '');
```

Opens a dialog box based on resource ID 1001 of (in this case) the default DLL, using the default dialog box procedure. The helpfile is provided by a `join_paths` command which produces a topic from the `acmanual` help folder.

### *echo*

```
echo(string)
```

Outputs *string* to the screen.

### *echo\_map*

```
echo_map(string)
```

As [echo](#) function, except that any nomenclature [mapping markers](#) in *string* are translated to the current nomenclature.

### Example

```
echo_map('#Cr# '.cr_number.' created')
```

Will echo into the output window, if the term CR has been mapped to RFC, 'RFC <cr number> created'

### *echo\_log\_action*

```
echo_log_action(message)
```

Appends *message* to the `acacts.log` file.

### *echo\_log\_error*

```
echo_log_error(message, fatal)
```

Appends *message* to the `acacts.log` file if (and only if) error logging is switched on — see [Configuration Options](#). If *fatal* is true the message is preceded by the string `ERROR:;` if it is false the message is preceded by the string `WARNING:.`

### *edit\_part\_perms*

```
edit_part_perms(partname)
```

Invokes the part permissions editing dialog on the specified part.

### *end\_progressbar*

```
end_progressbar()
```

Removes any existing progress bar from the screen.

This function is supported for both the Windows interface and the browser interface.

## *env\_ini\_var*

```
env_ini_var(env-var-name, registry-subkey)
```

Returns the value of the environment variable named *env-var-name* if it exists, otherwise the value of the registry entry in the *registry-subkey* specified in the subkey for the **AllChange** project .

If *intasoft.ini* files are in use instead of the registry then the *registry-subkey* is treated as the section of the *.ini* file. This follows the behaviour of **AllChange** when searching for most "environment variables".

### Example

```
env_ini_var('VISDIFFS', 'AllChange')
```

Might return C:\Program Files\Araxis\Araxis Merge v6.5\Merge.exe

## *env\_var*

```
env_var(env-var-name)
```

Returns the value of the environment variable named *env-var-name*.

### Example

```
env_var("SystemRoot")
```

Might return c:\WINDOWS

## *error*

```
error(message)
```

Causes a fatal error to be raised with *message* being issued as the error message, possibly together with other standard error message information. This will terminate the current operation.

If *message* is the empty string (' '), this raises a fatal error, just like normal, but without any error message/ box being displayed. This should not normally be used as this is not supported with the web interface to **AllChange**. Instead the user defined function **Cancel** should be used. This supports both the windows interface and the web interface. This takes a single parameter of an error message and will use error("") if used with the Windows interface and an error with the specified message in the web interface. **Cancel** supports nomenclature mappings. If invoked with an empty message then the standard message of 'not implemented in the web interface' will be issued.

Use **Cancel** after an error message has been displayed by some other means, e.g. a Visual ACCEL dialog or a DLL's own message, or after the user has pressed "Cancel" in response to a Visual ACCEL dialog.

### Example

```
if not file_exists(var(file)) then
    error("File does not exist (".var(file).")");
endif;
```

Will issue an error message if the file specified by the file user defined `ACCEL` variable does not exist and terminate the current operation

```
if message_box("Do you want to continue?", 'Continue', 4+16) != 6 then
    call(Cancel, 'Operation Cancelled');
endif;
```

If the user selects Cancel from the message box then the operation will be cancelled with the message 'Operation Cancelled' in the web interface and no message in the Windows interface

### ***error\_map***

`error_map(string)`

As [error](#) function, except that any nomenclature [mapping markers](#) in *string* are translated to the current nomenclature

#### **Example**

```
error_map('Cannot change to #part# outside current #workspace# (.pa_
part.')
```

Will issue an error message using the mapped nomenclature for the term part and the term workspace if there is one. If part is mapped to CI (configuration item) and workspace is mapped to the sandpit then the error message would read 'Cannot change to CI outside current sandpit (<config item name>'

### ***evaluate***

`evaluate(string)`

Evaluates *string* as an ACCELexpression and returns the result. Allows things like:

```
setvar(count, 1);
while var(count) <= 25 do
  echo('Arb field #'.var(count).' = '.
    evaluate('cr_arb'.var(count)));
  setvar(count, var(count)+1);
endwh;
```

### ***extract\_token***

`extract_token(var-name)`

Extract the next token from user-defined variable named *var-name*. Removes the token from the variable's value and returns it. Tokens are separated by whitespace; respects and removes quotes, like `split_unquote()`. Useful for parsing lines read from files.

#### **Example**

```
local(str);
setvar(str, 'abc def ghi');
extract_token(str)
```

Would return abc

### ***fclose***

`fclose(fhandle)`

Closes a previously opened (local or remote) file handle.

### ***fflush***

`fflush(fhandle)`

Flushes a file handle: output to a file is only guaranteed to be committed to the file on a `fflush()` or a `fclose()`. Useful if multi-user access to the file is on-going.

### *file\_browser\_reread*

```
file_browser_reread()
```

Causes the contents of the ACE File Browser to be reread. ACCEL functions which are known to change (local) files (e.g. `delfile()`, `takeout()`) automatically cause the File Browser to be reread. This function may be used explicitly if the File Browser needs rereading after an action, e.g. a `command()` is issued which creates a workfile.

### *file\_exists*

```
file_exists(filename)
```

Returns whether the file *filename* exists.

### *file\_exists\_in\_projdir*

```
file_exists_in_projdir(filename)
```

Returns whether the file *filename* exists in the project directory. *Filename* must be a plain filename (i.e. no path). This function determines whether the database is accessed locally or remotely under client/server operation.

#### **Example**

```
file_exists_in_projdir('ftpwspc.ac')
```

Would return true if the file existed in the project directory.

### *file\_exists\_in\_sysdir*

```
file_exists_in_sysdir(filename)
```

Returns whether the file *filename* exists in the system directory. *Filename* must be a plain filename (i.e. no path). This functions correctly whether the database is accessed locally or remotely under client/ server operation.

#### **Example**

```
file_exists_in_sysdir('report.sum')
```

Would return true if the file 'report.sum' exists in the system directory

### *file\_exists\_in\_templatedir*

```
file_exists_in_templatedir(filename)
```

Returns whether the file *filename* exists in the current project's template directory. *Filename* must be a plain filename (i.e. no path). This functions correctly whether the database is accessed locally or remotely under client/server operation.

#### **Example**

```
file_exists_in_templatedir('cr.tpl')
```

Will return whether a file named 'cr . tpl ' exists in the template directory for the current project

### *file\_locked*

`file_locked(filename)`

Returns whether file *filename* is "locked". This returns whether another process has the file open in a mode which denies write access (e.g. Microsoft Word).

### *filename\_part*

`filename_part(path)`

Returns the part whose related operating system path (file/ directory) name is given by *path*. The path must be a full operating system pathname (to a VC file/ directory). In addition the part can only be found when attached to a workspace such that the part is a descendent of the workspace's associated part. Neither the path nor the resulting part need exist.

#### **Example**

```
filename_part('c:-  
\allchange\acprojects\live\projdir\vcfiles\product\sw\source\vc\module1.c')
```

could return `/product/sw/source/module1.c` if the VC Files are stored in `c:\allchange\acprojects\live\projdir\vcfiles`.

### *file\_size*

`file_size(filename)`

Returns the size of the specified file

#### **Example**

```
file_size("C:\tmp\New Text Document.txt")
```

Might return 0

### *file\_suffix*

`file_suffix(filename)`

Returns the suffix (after the `.`, if any) on *filename*.

#### **Example**

```
file_suffix('part1.txt')
```

Returns `txt`

### *file\_time*

`file_time(filename)`

Returns the last modification time of file *filename* as the number of seconds elapsed since 1970/01/01 00:00:00 UTC.

#### **Example**

```
file_time('c:\temp\test1.ac')
```

Might return 1049211386

### *file\_writable*

`file_writable(filename)`

Returns whether the file *filename* is writable (i.e. not read-only).

#### **Example**

```
file_writable('c:\temp\test1.ac')
```

Returns true if the file `c:\temp\test1.ac` is writable, i.e. does not have the read only attribute set

### *find\_executable*

`find_executable(filename)`

Returns the File Manager/ Explorer association for *filename*.

#### **Example**

```
find_executable('c:\temp\test1.ac')
```

Might return `C:\dev\ac\bin\win32\Acredit.exe`, as the program associated with `'c:\temp\test1.ac'` to open the file

### *flock*

`flock(fhandle, lock)`

Acquires/ releases an exclusive lock on a file handle according as *lock* is true or false. Useful to prevent multi-user interference.

### *fopen*

`fopen(filename, access)`

Opens a file for i/o. *Filename* is the file name/ path; *access* should be one of 'r' (read), 'w' (write) or 'a' (append). Returns a file handle (a number) which should be stored in a variable and passed to all other function calls accessing the file. It is the programmer's responsibility to close the file when no longer needed; there are a limited number of file handles available simultaneously. *Passing an invalid file handle to another function results in undefined (possibly nasty!) behaviour.* An error is raised if the file cannot be opened.

#### **Example**

File handling (append one file to another, locking out other users from doing the same):

```
setvar(fin, fopen('infile', 'r'));
setvar(fout, fopen('outfile', 'a'));
flock(var(fout), true);
setvar(line, freadline(var(fin)));
while var(line) != '<EOF>' do
    fwriteline(var(fout), var(line));
    setvar(line, freadline(var(fin)));
endwh;
fflush(var(fout));
flock(var(fout), '');
fclose(var(fout));
fclose(var(fin));
```

### *fopen\_in\_projdir*

`fopen_in_projdir(filename, access)`

Opens *filename* (which must be a relative filename, i.e. no path or a path relative to the project directory) in the project directory for i/o and returns a handle, cf. [fopen\(\)](#). Access may be any of the modes accepted by [fopen\(\)](#). This function works whether the project directory is accessed locally or remotely under client/server operation. Note that this means that files can be created and updated in the project directory even when running client/server.

All other file handling functions ([freadline\(\)](#), [fwriteline\(\)](#), [fclose\(\)](#) etc.) work whether the file is accessed locally or remotely under client/server operation. File buffering is performed at the client side, so remember that file writes are only guaranteed to be committed on a [fflush\(\)](#) or [fclose\(\)](#).

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

### *fopen\_in\_sysdir*

`fopen_in_sysdir(filename, access)`

As [fopen\\_in\\_projdir\(\)](#), but gives access to files in the **AllChanges** system directory when running client/server.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

### *fopen\_in\_templatedir*

`fopen_in_templatedir(filename, access)`

As [fopen\\_in\\_projdir\(\)](#), but gives access to files in the current project's template directory when running client/server.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

#### **Example**

```
setvar(fpin, fopen_in_templatedir('cr.tpl', 'r'))
```

Will open the file 'cr.tpl' in the projects template directory for read access and assign the file handle to the variable `fpin`

### *format\_date*

`format_date(date-string, format)`

Returns a string containing the required portion specified by *format* of the date/ time given as *date-string*.

The date must be a string of the form: `yyyy/mm/dd hh:mm:ss`. The format may be any of those described under [sys\\_date](#).

#### **Example**

```
format_date(pa_date, 'winlongdate')
```

Might return 21 December 2003

### *format\_number*

`format_number(<number>)`

Converts a string in the Intasoft internal format of `<num>.<num>` to the local format for numbers.

**Example**

`format_number("1.23")` will produce "1,23" for some countries and "1.23" for others as appropriate

***freadline***

`freadline(fhandle)`

Returns the next line from a (local or remote) file handle. The string <EOF> is returned on end-of-file.

See [fopen](#)

***full\_to\_username***

`full_to_username(fullname)`

Converts a users full name as defined in the user registrations to their login name/ID. If no full name has been specified for the user, the name passed is returned.

**Example**

If the user registrations define that user name 'fred' has a full name of 'Fred Bloggs' then:

`full_to_username('Fred Bloggs')` returns 'fred'

***full\_to\_usernamelist***

`full_to_usernamelist(realnamelist)`

Converts a list of full/friendly names as defined in the user registrations to user login name/IDs. If no full name has been specified for a user, the name passed is returned.

**Example**

If the user registrations define that user name 'fred' has a full name of 'Fred Bloggs' and user named jon has the full name of 'Jonathan Smith' then:

`full_to_usernamelist("'Fred Bloggs' 'Jonathan Smith'")`

Returns fred jon

***fwriteline***

`fwriteline(fhandle, str)`

Writes a string, followed by a newline, to a file handle.

See [fopen](#)

***fwritestr***

`fwritestr(fhandle, str)`

Writes a string to a file handle.

See [fopen](#)

## *getarg*

`getarg(arg)`

Returns the value, if any, of the argument named *arg* in the current command; this is very useful in writing command conditions and actions. If the argument has been used with no value the value `yes` is returned; if the argument has not been used at all the value `' '` is returned. *Arg* should be of the form `-argument` if the argument is shown with a leading `-` in the manual, and of the form `argument` if it is shown as `argument=value` in the manual.

*Arg* may be a list of arguments. The function returns the value of the first argument in the list which has been used (or `' '` if none). This is useful when checking whether an arbitrary field has been specified by either its proper or aliased name, see also [getarg\\_arb](#).

### Example

Thus if the user issues the command:

```
somecommand field=value -option1 something -option2
```

`getarg()` used in the conditions or actions for `somecommand` would return the following values:

```
getarg('field') => 'value'
getarg('-option1') => 'something'
getarg('-option2') => 'yes'
```

and any other `getarg()` would return `' '` (i.e. false).

```
setvar(temp, getarg('priority arb2'))
```

would — inside, say, `altercr` — pick up both `altercr priority=...` and `altercr arb2=...`

## *getarg\_arb*

`getarg_arb(arbfieldname, class)`

Returns the value, if any, of the specified arbitrary field argument to the current command. This is like a specialised `getarg()` just for arbitrary fields. *Arbfieldname* must name the real arbitrary field name (and so be quoted), e.g. `"pa_arb1"`; *class* should name the class to which the arbitrary field applies, or `any` if it applies to any class. Unlike `getarg()`, `getarg_arb()` recognises an assignment on the command-line to an arbitrary field by either its real name or its aliased name.

### Example

```
altercr "arb1=High Priority"                                or
altercr "Priority=High Priority"                            =>
getarg_arb("cr_arb1", cr_class) == "High Priority"
```

## *get\_branch\_list*

`get_branch_list(part)`

Returns a list of all the branch names valid for *part* as defined in the [branch name definitions](#). *part* must be the full path to a part. Note that this does not include branch names which are not listed in the branch name definitions.

### Examples

`get_branch_list('/product/software')` will return the list of branch names which have been defined and are valid for `/product/software`.

***get\_cnum\_resource\_value***

```
get_cnum_resource_value(resnum)
```

Returns the value of the CR numbering resource associated with *resnum*. *Resnum* is a digit between 0 and 9, corresponding to the [counter](#) specified in the CR class definition. The value returned is the last CR number generated for the specified counter.

The resource/counter may be reset using [set\\_cnum\\_resource\\_value\(\)](#).

**Example**

```
get_cnum_resource_value(1)
```

Might return 100 if SCR00100 is the last CR created using the first CR numbering counter (used by the SCR class in the [Intasoft Standard](#) out-of-the-box configuration).

***get\_first\_voter\_for\_user***

```
get_first_voter_for_user(database, item, status, user)
```

Returns the first defined voter for the vote specified by the *database*, *item* and *status* which includes the specified user. This function is used by the email submission of votes for determining the voter represented by a user. This might be Role, Group, Field or individual user.

**Example**

```
get_first_voter_for_user('cr' 'DCR00010', 'Review', user)
```

Returns the voter that the current user is representing in a vote for DCR00010 in the Review status

***get\_next\_item***

```
get_next_item(database, item)
```

Returns the name of the item that appears immediately after the specified item in the database specified by *database* using the primary index. This is only valid for the CR and Baseline databases. All other databases will return an empty string.

**Examples**

```
get_next_item('cr', 'SCR00005') will return SCR00006
```

***get\_part\_filter***

```
get_part_filter()
```

Returns the current value of the Filter selection in the Part Browser or ' ' if the part browser is not the front window.

Note that the value will be as shown in ACE taking account of any [nomenclature mappings](#).

This function is only implemented from ACE.

***get\_prev\_item***

```
get_prev_item(database, item)
```

Returns the name of the item that appears immediately before the specified item in the database specified by *database*. This is only valid for the CR and Baseline databases. All other databases will return an empty string.

**Examples**

`get_prev_item('cr', 'SCR00006')` will return SCR00005

***get\_selected***

`get_selected(type)`

Returns a space-separated string of items currently selected in the *front* browser/ viewer window, provided that window has items of an appropriate type. Type can be one of:

<b>any</b>	gets selections from the current list on the current window.
<b>File</b>	gets selections from files browser.
<b>CR</b>	gets selection from CR browser or viewer.
<b>Baseline</b>	gets selections from baselines browser or viewer.
<b>Attachment</b>	gets selections from Relationships tab when Attachments are selected in CR or baselines viewer.
<b>Part</b>	gets selections from parts viewer or parts, baseline details or CR/ partsaffected browser.
<b>Issue</b>	as for <b>Part</b> , or from check-outs browser or viewer.
<b>Vote</b>	gets selections from votes browser or viewer or votes tab of CR, part or baseline viewer. This returns a list of votes each of which is tab separated in the form:  <pre>type tab item tab date/time tab user</pre> Where the <i>type</i> is cr, part or baseline and <i>item</i> is the cr, part or baseline on which the vote has been cast.
<b>Instance</b>	gets selections from the instance browser or viewer or the Instances tab of the part viewer.
<b>Monitor</b>	as for <b>Issue</b> , or from monitor or files browser or CR viewer.
<b>Role</b>	gets selections from roles browser.
<b>Workspace</b>	gets selections from workspace browser.

The empty string is returned if the up-front window is not of an appropriate type (or if no items are selected).

This function may be used to allow user-defined functions to act on whatever item(s) have been selected from a suitable window, just like predefined **AllChange** commands. The user is expected to up-front an appropriate window, select some items, and then choose the user-defined function from the **Functions** menu.

This function is only implemented from ACE.

***get\_selected\_issue***

`get_selected_issue()` Returns a space-separated string of items currently selected in the issue/check-out browser, issue/check-out viewer or the issues/check-outs tab on the part viewer. Unlike `get_selected()`, each item is of the form:

```
partname tab workspacename
```

***get\_selected\_list***

`get_selected_list()`

Returns the name of the current list (i.e. the list with the current focus).

***get\_selected\_list\_db***

`get_selected_list_db()`

Returns the name of the database associated with the current list (i.e. the list with the current focus).

### *get\_selected\_part*

`get_selected_part(versions)`

Returns a space-separated string of parts currently selected in the part browser/ viewer window. If *versions* is true then selected versions only will be returned; if it is false then selected components/ sub-systems only will be returned; if it is *any* then anything selected will be returned.

This function is only implemented from ACE.

### *get\_system\_flag*

`get_system_flag(flag)`

Returns true/ false according as the system flag is currently switched on/ off. *Flag* may be any of the flag names accepted by the **set** command.

#### **Example**

```
get_system_flag('actions')
```

Returns true if the Actions system flag is on and false if it is off

### *get\_upfront\_window*

`get_upfront_window()`

Returns the name of the currently up-front window. This will be one of: Part Browser, Part Viewer, Issue Browser, Issue Viewer, File Browser, Monitor Browser, CR Browser, CR Viewer, Baseline Browser, Baseline Viewer, Baseline Detail Viewer, Instance Browser, Instance Viewer, Role Browser, Workspace Browser, Output, Find, Vote Viewer, or '' if none.

Note that the name returned may not be the same as that shown in ACE if [nomenclature mappings](#) are in use; for example out-of-the-box maps Issue to Check-out.

This function is only implemented from ACE.

### *get\_window\_field*

`get_window_field(window, field)`

Returns the value of a field as currently displayed in a viewer window or add/alter dialog.

This differs from the actual database field in that the viewer fields can be changed without committing those changes to the database. Using this function allows ACCEL code to determine what a user has entered in a field in order to present the user with, say, appropriate options for another field's values (e.g. when such a field has a "... button).

The *window* parameter should specify a window name in the form returned from [get upfront window](#), in addition *<Database> Vote Viewer* may be specified to denote a specific Vote Viewer. When `get_window_field` is used and a dialog is open which is appropriate for the specified viewer then the function will get the field value from the dialog.

The dialogs which will be used for particular viewers are as follows:

**CR Viewer:** Add CR, Alter CR

**Part Viewer:** Add Part, Alter Part, New Version, Alter Version

**Baseline Viewer:** Add Baseline, Alter Baseline

**Vote Viewer:** Cast Vote

(All other dialogs do not show fields which can have their value expressions defined, and so are not relevant.)

When `get_window_field` is called, **AllChange** will check to see if there is an Add or Alter dialog open which is suitable for the database used by the specified viewer.

If there is no dialog open, then **AllChange** will look for the specified viewer window. If there is more than one matching viewer open then the top-most matching viewer will be used.

If no suitable dialog is open, or there is no matching viewer, the function will return the empty string. If an invalid *window* parameter is specified an error will be given.

The *field* parameter should specify the 'old' name for viewer's fields, e.g. 'pa\_arb1' not 'pa\_Description'. In addition to arbitrary fields, the fields which may be specified are those which appear on the General, Header, Part, Version and Text tabs of the relevant viewer.

Fields that do not appear on Add/Alter dialogs, and are not directly editable in viewers, e.g. 'cr\_assignee' may not be used with `get_window_field`. Also, information on dialogs that does not correspond directly to a database field cannot be returned with the function, e.g. 'Header Only' on the 'Add Baseline' dialog.

Note that if a field is requested that is not user-editable in viewers, for instance, 'cr\_class', then the function will return the value from the database record held in memory by the viewer.

Specifying a window and/or a field which is not valid for the function will produce an error.

In order to use the `get_window_field` function in an expression for an arbitrary field's value list, the field should use a single- or multi-select list rather than a drop-down list, so that the expression is re-evaluated. Drop-down list expressions are only evaluated when initially populating the viewers/dialogs.

### Example

`get_window_field('CR Viewer', 'cr_summary')` will return the value currently shown in the edit control for the Summary in the Add CR, Alter CR dialogs or the CR Viewer window depending on what is open/up front. If none of these are open, then the empty string is returned

### *getserverfile*

`getserverfile(fromfile, tofile)`

If running client/server gets (copies) a file from the server to the client; *fromfile* should specify a path from the server's point of view while *tofile* should specify a path from the client's point of view. The file's permissions and timestamp are retained. If not running client/server acts the same as `copyfile()`.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

### Example

```
getserverfile(join_server_paths(acserverprojdir, 'test1.ac'),
'c:\temp\test1.ac')
```

Copies the file test1.ac from the server's project directory to the client's c:\temp directory

### *getservertime*

`getservertime():`

If running client/server, returns the current time *on the server* as a number of seconds elapsed since 1970/01/01 00:00:00 UTC (GMT). If not running client/server, returns the time on the client, cf. `gettime()`.

### Example

```
seconds_to_date(getservertime())
```

## *gettime*

`gettime()`:

Returns the current time *on the client* as a number of seconds elapsed since 1970/01/01 00:00:00 UTC (GMT).

### **Example**

```
seconds_to_date(gettime())
```

## *get\_window\_item*

`get_window_item(window)`

Returns the item currently displayed in the specified window, if any. Possible values for *window*, and what is returned, are as follows:

<b>Part Viewer</b>	the name of the part currently shown in the part viewer
<b>Version Viewer</b>	the name of the version currently shown in the part viewer
<b>CR Viewer</b>	the number of the CR currently shown in the CR viewer
<b>Baseline Viewer</b>	the name of the baseline shown in the baseline viewer
<b>Baseline Detail Viewer</b>	the name of the baseline shown in the baseline detail viewer, a tab character and then then name of the detail shown in the viewer
<b>Issue Viewer</b>	the part/ version currently shown in the check-out/issue viewer
<b>Issue Workspace Viewer</b>	the workspace of the check-out shown in the check-out/issue viewer
<b>&lt;Database&gt; Vote Viewer</b>	the database, item, date and user of the vote currently shown in the vote viewer for <Database> separate by tabs. <Database> may be CR, Part or Baseline
<b>Part Browser</b>	the subsystem currently open in the part browser (same as <code>cwp</code> )
<b>File Browser</b>	the directory currently open in the file browser

In order to use the *get\_window\_item* function in an expression for an arbitrary field's value list, the field should use a single- or multi-select list rather than a drop-down list, so that the expression is re-evaluated. Drop-down list expressions are only evaluated when initially populating the viewer.

## *has\_perm*

`has_perm(part, role)`

Returns whether the current user has permission to have the role *role* for the part *part*. If *part* is empty returns whether the current user has permission to have *role* on *any* part.

In order to have permission: if the user has not set his current role he must be a valid user for the specified role for the specified part, otherwise his current role must match the specified role.

There are however two roles which *has\_perm* treats specially: `dbadmin` and `dbsuperuser`. If the user has `dbadmin` role on a part he is granted all permissions on that part with the single exception of

`dbsuperuser` permission; if the user has `dbsuperuser` role on a part he is granted all permissions on that part with no exception.

Like any other role, these are assigned to users over areas of the parts tree and a user may only adopt one of these roles provided he has been assigned that role over at least one area of the parts tree. As a safety precaution, whereas a user who has not explicitly adopted any role is granted permission for any normal role to which he is entitled on a particular part, `dbadmin` and `dbsuperuser` permissions will only be granted if the user has explicitly set his current role to one of these.

### Example

```
has_perm('/', 'dbsuperuser')
```

Returns whether the current user has `dbsuperuser` permission on /

### *has\_role*

```
has_role(user, part, role)
```

Returns whether the user *user* is allowed the role *role* for the part *part*. If *part* is empty returns whether *user* is allowed *role* on *any* part.

### Example

```
has_role('fred', '/part', 'developer')
```

Returns whether the fred has the developer role on /part

### *ifelse*

```
ifelse(condition, if-result, else-result)
```

Tests *condition*. If it is true returns *if-result*, else returns *else-result*. Reduces the need for *if-then-else* statements.

```
yn(cond) == ifelse(cond, 'yes', 'no')
```

### Example

```
ifelse(pa_obsolete, "no longer used", "still in use")
```

### *index\_names*

```
index_names(database)
```

Returns of the names of all the fields which may be used to index the database *database*. *Database* must identify a real database, like one passed to [num\\_records\(\)](#). Each name in the returned list looks like the string that might be passed as the argument to `-indexedby to report/each`, See also [display\\_index\\_to\\_real\\_index\(\)](#).

### Example

```
index_names('cr')
```

Would return number obsolete class status date toppart reference originator assignee summary arb1 arb2 ... arb40, i.e. all the fields in the actual CR database.

***ini\_var***

```
ini_var(var-name, registry-subkey)
```

Retrieves the value of the specified *var-name* of the registry entry in the *registry-subkey* specified in the subkey for the **AllChange** project .

If *intasoftini* files are in use instead of the registry then the *registry-subkey* is treated as the section of the .ini file.

**Example**

```
ini_var('ACPROJECT', 'AllChange')
```

Might return MyProject

***instance\_list***

```
instance_list(version)
```

Returns a list of the instances for the specified *version*.

**Example**

```
instance_list('/Car/CD Player;004')
```

Returns a list of the instances of version 004 of /Car/CD Player. If there are 2 instances then this would be:

```
/Car/CD Player;004:00001
```

```
/Car/CD Player;004:00002
```

***instance\_part***

```
instance_part(instance)
```

Returns the part with the instance number removed from *instance*.

**Example**

```
instance_part('/Car/CD Player;004:0023')
```

Returns /Car/CD Player;004

***instr***

```
instr(str1, str2)
```

Returns the position of the first occurrence of *str2* in *str1*, or -1 if not found. The position counts from 0.

**Example**

```
instr('0123456789012', '1')
```

returns 1

```
instr('0123456789012', 'z')
```

returns -1

***interpret***

```
interpret(command)
```

Performs an **AllChange** command which is given as the string *command*. The command should be in the form of the **AllChange** command line required to invoke the command. This function is widely used in writing command actions.

The default behaviour of `interpret()` when a fatal error occurs in the course of interpreting a command is that the error is propagated back to whatever called the `interpret()` causing it to exit, until the top level is reached. However, if *command* begins with a - (minus) the `interpret()` instead returns whether it "succeeded", i.e. did not result in an error. This allows errors occurring during `interpret()` to be ignored, or the success/ failure of an `interpret()` to be tested in an `if ...` statement and desired action taken appropriately.

### *is\_app\_running*

`is_app_running(app-name)`

Returns whether the named application is currently running. *App-name* should match the *beginning* of an application window name, as it appears in the title bar or Task List. It is not necessary to specify the complete title, so, for example, if there is a window named **Microsoft Word - Document1** then `is_app_running('Microsoft Word')` will return true.

### *is\_blocked\_vote*

`is_blocked_vote(database, item, status)`

Returns whether the most recent vote set in *database* for *item* in *status* resulted in an inconclusive decision.

See also [is\\_open\\_vote](#)

## Example

```
is_blocked_vote('cr', 'RFC00010', 'CAB')
```

Might return true if the last vote on RFC00010 in the CAB status resulted in a blocked vote

### *is\_command*

`is_command(command)`

Returns whether the named command is a true **AllChange** command (e.g. `add`, `newcr`), as opposed to a user-defined function (e.g. `CheckIn`).

### *is\_def\_version*

`is_def_version(version)`

Returns whether the version is the default version of its component. The version should be specified as a full path, including the version specifier. The default version of a part is generally the top version, unless the registrations have modified this.

## Example

```
is_def_version('/product/sw/source/module1.c;004')
```

Returns whether 004 is the default version of `/product/sw/source/module1.c`

### *is\_field\_readable*

`is_field_readable(database, fieldname, item, usertypeonly, ignoresecurecontext)`

Returns whether a field or item is readable by the current user with their current role. If read access is allowed then **true** is returned otherwise **false** is returned.

*database* must be one of `cr`, `part`, `baseline`, `version`, `part_status_log`, `baseline_status_log` or `cr_status_log`.

*fieldname* may be the name of the field without its database prefix, e.g. `summary`, `arb1` or it may be `any`. If blank then `any` is assumed.

*item* may be a part (full part path), `cr` (number) or baseline (name) or it may be `any`. If blank then `any` is assumed.

*usertypeonly* should be true or false. If true is specified then the function will return whether the field is readable based on the type of the user only and not on their role.

*ignoresecurecontext* should be true or false. This allows code executed within a secure context to determine whether a field would be readable outside of the secure context. This parameter only applies when executed in a secure context. Note though that if the user's current role is set to `dbsuperuser`, then `is_field_readable` will always return **true**.

### Examples

```
is_field_readable('cr', 'Priority', cr_number, false, false)
```

This will return whether the Priority arbitrary field is readable by the current user/role for the CR which is the current record.

### *is\_group\_user*

```
is_group_user(groupname, username)
```

Returns true if *username* is *groupname*, or a member of the group named by *groupname*.

### Example

```
is_group_user('developers', 'fred')
```

Returns whether fred is a member of the developers group

### *is\_in\_baseline*

```
is_in_baseline(part-or-baseline, baselinename)
```

Return whether the part or baseline *part-or-baseline* is contained in the baseline named *baselinename*. If *part-or-baseline* starts with / it is taken as a part, otherwise it is taken as a baseline name. A part must be given as a full path name; if it ends in a ; the function returns whether *any* version of the part appears in the baseline. If *baselinename* is a meta-baseline its constituent baselines are followed down recursively trying to find *part-or-baseline*. If *baselinename* is `any` the function returns whether the *part-or-baseline* is contained in any baseline.

### Example

```
is_in_baseline('/product/sw/source/module1.c;', 'release')
```

Returns whether any version of /product/sw/source/module1.c is in the baseline named release

```
is_in_baseline('release', 'meta')
```

Returns whether the baseline named release appears in the meta-baseline named meta (including via any sub-meta-baselines which meta might include)

```
is_in_baseline('/product/sw/source/module1.c', any)
```

Returns whether any version of /product/sw/source/module1.c is in any baseline

### *is\_in\_cr\_partsaffected*

`is_in_cr_partsaffected(cr, part)`

Returns whether *part*, or any version of *part*, is among the specified *cr*'s *partsaffected*. This can be much more efficient than searching through *cr\_partsaffected*.

#### **Example**

```
is_in_cr_partsaffected('CR00001', '/product/sw/source/module1.c')
```

Returns whether `/product/sw/source/module1.c`, or any version of `/product/sw/source/module1.c`, is among CR00001's *partsaffected*

### *is\_in\_group\_list*

`is_in_group_list(username, grouplist)`

Returns `true` if user is in *grouplist*, or is a member of a group named by *grouplist*.

#### **Example**

```
is_in_group_list('fred', 'developers managers')
```

Returns whether fred is a member of the developers or managers groups

### *is\_in\_list*

`is_in_list(str, list)`

Returns whether the string *str* is in the list *list*.

#### **Example**

```
is_in_list('two', 'one two three')
```

Returns `true`

### *is\_instance*

`is_instance(part)`

Return whether the part *part* refers to an instance.

#### **Example**

```
is_instance('/Car/CD Player;004:0023')
```

Returns `true`

### *is\_issued*

`is_issued(part, workspace, uname, foredit)`

Returns whether the part *part* is checked out to the workspace named *workspace* and the user named *user*. Both *workspace* and *user* may be specified as `any` indicating any workspace or any user.

If the *foredit* parameter is `true` then the part must be checked out for edit. If it is `false` then the part must not be checked out for edit. If it is `any` then the part may be either checked out for edit or for read only.

The part must be a full path name for the part.

**Example**

```
is_issued('/product/sw/source/module1.c', workspace, user, true)
```

Returns whether `/product/sw/source/module1.c` is checked out for edit to the current user in the current workspace

```
is_issued('/product/sw/source/module1.c', any, any, any)
```

Returns whether `/component` is checked out for any purpose to any user in any workspace

***is\_number***

```
is_number(str)
```

Returns whether *str* is an integer number.

**Example**

```
is_number('123')
```

Returns `true`

***is\_open\_vote***

```
is_open_vote(database, item, status)
```

Returns whether the *item* in *database* in *status* has a current open vote.

See also [is\\_blocked\\_vote](#)

**Example**

```
is_open_vote('cr', 'RFC00010', 'CAB')
```

Might return `true` is an open vote on RFC00010 in the CAB status

***is\_out\_for\_edit***

```
is_out_for_edit(version)
```

Returns whether *version*, which should be the full path of a version, is currently out for edit, i.e. has a successor which is checked out for edit.

**Example**

```
is_out_for_edit('/product/sw/source/module1.c;004')
```

Returns whether `/product/sw/source/module1.c;004` is checked out for edit to anyone

***is\_project\_user***

```
is_project_user(user)
```

Returns whether the specified *user* is a user for the current project. *user* should be the logon name for the user in question.

## Examples

`is_project_user(fred)` returns whether 'fred' is a valid user for the current project.  
`is_project_user(full_to_username("Fred Bloggs"))` returns whether 'Fred Bloggs' is a valid user for the current project.

### *issued\_version*

`issued_version(part, workspace, uname, foredit)`

Returns what version, if any, of a part has been checked out. Arguments are as for [is\\_issued\(\)](#).

#### Example

```
issued_version('/product/sw/source/module1.c', 'workspace1', any, any)
```

Would return `/product/sw/source/module1.c;004` if that version is checked out to workspace1

### *is\_version*

`is_version(part)`

Return whether the part *part* refers to a version.

#### Example

```
is_version('/product/sw/source/module1.c;004')
```

Returns `true`

### *is\_voter\_mandatory*

`is_voter_mandatory(database, item, status, user)`

Returns whether the specified *user* is mandatory for a vote defined for the specified *database*, *item* and *status*.

This function is deprecated and provided for backward compatibility, use [is\\_voter\\_optional](#) instead.

#### Example

```
is_voter_mandatory('cr', 'RFC00123', 'CCB', user)
```

Returns whether the current user is a mandatory voter for RFC00123 when it is in the 'CCB' status.

### *is\_voter\_optional*

`is_voter_optional(database, item, status, user)`

Returns whether the specified *user* is an optional voter for a vote defined for the specified *database*, *item* and *status*. It returns true if the *user* is optional in all of the defined voters, i.e. if the user is named in a group that is optional, and also named in a field which is not, then the function will return false.

#### Example

```
is_voter_optional('cr', 'RFC00123', 'CCB', user)
```

Returns whether the current user is an optional voter for RFC00123 when it is in the 'CCB' status.

***is\_workspace\_user***

```
is_workspace_user(user, workspacename)
```

Returns whether the user named *user* is a valid user of the workspace *workspacename*.

**Example**

```
is_workspace_user
  is_workspace_user('fred', 'workspace1')
```

Returns whether fred is a valid user of workspace workspace1

***join***

```
join(list, joiner)
```

Returns the string resulting from joining each element of *list* to the next with string *joiner*. When ACCEL automatically converts a list to a string this is equivalent to `join(list, ' ')`.

**Example**

```
join('1 2 3', '->')
```

Returns 1->2->3

***join\_instance***

```
join_instance(version, instance)
```

Returns the instance resulting from joining *instance* onto the end of *version*; if *version* already has an instance on the end it is replaced by *instance*.

This function just treats *version* and *instance* as strings: it does not access the database or require that they actually exist.

**Example**

```
join_instance('/Car/CD Player;004', '0023')
```

Returns /Car/CD Player;004:0023

***join\_parts***

```
join_parts(part1, part2)
```

Returns the part resulting from joining *part2* onto the end of *part1* (if *part2* is absolute then returns *part2*; squeezes leading ./ and/ or ./ out of *part2*).

**Example**

```
join_parts('/subsys1', 'subsys2/component')
```

Returns /subsys1/subsys2/component

```
join_parts('/anything', '/subsys2/component')
```

Returns /subsys2/component

### *join\_paths*

`join_paths(path1, path2)`

Returns the path resulting from joining operating system path *path2* onto the end of *path1* (if *path2* is absolute then returns *path2*).

#### **Example**

```
join_paths('\\server\share', 'dir')
```

Returns \\server\share\dir

```
join_paths('\\anything', 'c:\dir')
```

Returns c:\dir

### *join\_version*

`join_version(part, version):`

Returns the version resulting from joining *version* onto the end of *part*; if *part* already has a version on the end it is replaced by *version*.

This function just treats *part* and *version* as strings: it does not access the database or require that they actually exist.

#### **Example**

```
join_version('component;001', '004')
```

Returns component;004

### *joinquote*

`joinquote(list, joiner)`

Like [join\(\)](#), but quotes elements when building the string if necessary. Use `joinquote()` if *list* holds filenames which may contain spaces and thus need to be quoted.

#### **Example**

```
join(splitunquote('a "b c" d', ' '), ' ') → 'a b c d'
```

```
joinquote(splitunquote('a "b c" d', ' '), ' ') → 'a "b c" d'
```

### *join\_server\_paths*

`join_server_paths(path1, path2)`

If running client/server gets the server to return the path resulting from joining *path2* onto the end of *path1*; *path1* and *path2* should specify paths from the server's point of view. This function should be used to produce paths suitable for passing to functions which require paths from the server's point of view (`createserverdir()`, `getserverfile()` etc.), especially if the client and server are running different operating systems. If not running client/server acts the same as `join_paths()`.

#### **Example**

```
join_server_paths(acserversysdir, 'win32')
```

Might return `c:\acsys\win32`

### *legal\_dos\_pathname*

`legal_dos_pathname (pathname)`

Returns whether *pathname* is a legal MS-DOS/ Windows 3.x pathname, i.e. no segment must exceed 8.3 characters. Used (only) if the **Check DOS Filenames** option is selected in ACCONFIG.

### *listlen*

`listlen (lst)`

Returns the number of items in list *lst*.

### **Example**

```
listlen('one two three')
```

Returns 3

### *local*

`local (var {, var})`

Creates the specified local user-defined variable(s). Variables are local to the block (condition, action, or image section of a report format file) in which the `local` appears. Any current value for the variable is saved when the `local` is met and restored on exit from the block. The implementation is such that any functions called from the block will see only the local variable (not any outer instantiation), though it would be wise not to rely on this behaviour.

### **Example**

```
local (var1, var2, var3)
```

Defines 3 local variables

### *localtime\_to\_utc*

`localtime_to_utc (date-time)`

Converts *date-time* from local time to UTC (GMT).

*Date-time* should be specified in standard Intasoft format (YYYY/MM/DD hh:mm:ss).

The string returned is in standard Intasoft format.

### **Example**

```
localtime_to_utc("2002/10/10 19:00:00")
```

Will return "2002/10/11 00:00:00" if you are in EST

### *lowercase*

`lowercase (str)`

Returns string *str* converted to lower case.

### Example

```
lowercase('ABCdef')
```

Returns abcdef

### *make\_bt\_file*

```
make_bt_file(workfile, version, edit)
```

Makes (i.e. writes) the Build Thread file corresponding to *workfile*. *Workfile* should name the (absolute or relative) path to the workfile; *edit* should be true if the workfile was taken out for edit, false otherwise; *version* should name (the full path of) the version which was taken out to produce *workfile* if *edit* is false, or it should name the new reserved version if *edit* is true.

This function should be used whenever a BT is to be generated, whether in the actions for the **issue** command or for a workfile that was taken out directly rather than through the normal check out mechanism; care should be taken that the version specified really was the one from which the workfile was produced.

### *make\_user\_list*

```
make_user_list(type, name, database, item)
```

Returns a list of users matching the specified *type* and *name*, for the specified *database* and *item*. The type may be one of **User**, **Role**, **Field** or **Voters**, with *name* specifying the user, role or field, or, for Voters, "(Voted)", "(Not Voted)" or "(All Voters)".

When *type* is **Voters**, the current or latest vote set is used.

The *user* should be specified as the registered user name (i.e. the log in ID),

The *field* should be specified as the Display name for the field as specified in [Field Name Definitions](#).

The *database* and *item* are used to determine the correct role, field value or vote set to use.

### Example

```
make_user_list('Role', 'crmanager', 'cr', 'SCR00123')
```

Returns a list of users with the crmanager role for CR SCR00123

```
make_user_list('Voters', '(Not Voted)', 'cr', 'SCR00123')
```

Returns a list of users which have not voted on SCR00123.

### *map\_nomenclature*

```
map_nomenclature(string)
```

Translates text containing nomenclature [mapping markers](#) to the current nomenclature, and returns the mapped text.

### Example

```
map_nomenclature('You have been assigned #Cr# ')
```

Will return 'You have been assigned RFC ' if the term CR has been mapped to RFC

## *match\_wild*

`match_wild(str, pattern)`

Return whether the string *str* matches the pattern specified by *pattern*.

In the pattern a "?" matches a single character and a "\*" matches zero or more characters.

### **Example**

```
match_wild('abcdef', 'a?c*d*')
```

Returns `true`

## *message\_box*

`message_box(text, caption, style)`

Displays a Windows message box with the specified text, caption and style. This function maps directly to the Windows `MessageBox` function. Multi-line text is permitted by embedding the Windows newline characters, "`^r^n`", in *text*.

*Style* is a number with any of the desired `MessageBox` styles; this is expressed as a combination of ACCEL variables, and values may be:

Variable	Value	Buttons
<code>mb_ok</code>	0	Ok Button
<code>mb_okcancel</code>	1	Ok, Cancel Buttons
<code>mb_abortretryignore</code>	2	Abort, Retry, Ignore Buttons
<code>mb_yesnocancel</code>	3	Yes, No, Cancel Buttons
<code>mb_yesno</code>	4	Yes, No Buttons
<code>mb_yestoall</code>	16777216	Yes, Yes to All, No, Cancel buttons
<code>mb_notoall</code>	33554432	Yes, No, No to All, Cancel buttons

In addition if an Icon is desired in the message box the following values may be added to the button values shown above:

Variable	Value	Icon
<code>mb_icon_stop</code>	16	Hand/Warning Icon
<code>mb_icon_question</code>	32	Question Icon
<code>mb_icon_warning</code>	48	Exclamation Icon
<code>mb_icon_info</code>	64	Asterisk/Information Icon

In addition `MB_TASKMODAL` is automatically added in (meaning that ACEwindows are disabled until the user responds to the message box).

Returns the button pressed to exit the message box as a numeric. ACCEL variables are defined to ease programming.

Variable	Value	Button
<code>mb_btn_ok</code>	1	Ok
<code>mb_btn_cancel</code>	2	Cancel
<code>mb_btn_abort</code>	3	Abort
<code>mb_btn_retry</code>	4	Retry

mb_btn_ignore	5	Ignore
mb_btn_yes	6	Yes
mb_btn_no	7	No
mb_btn_yestoall	100	Yes to All
mb_btn_notoall	101	No to All

## Web Browser Interface

There is an HTML based version of this function which will be used when the function is invoked from the browser interface. The HTML version of the function is called `web_message_box` and is defined in the ACCELfunction file `webifunc.ac`. You do not need to call the function directly, as the implementation of `message_box` will call it when run in the browser interface..

Not all the options are accepted by the web implementation. Icons are not used, and so the icon options are ignored. The OK and OK/Cancel options have direct equivalents, while the Yes/No option is shown as OK/Cancel. The remaining options are not implemented, and will produce an error message.

## Example

```
message_box("Don't do that!", "Error", mb_ok + mb_icon_stop)
```

displays a message box with an **OK** button and a **Warning** icon;

```
if message_box("Continue?", "Question", mb_yesno + mb_icon_question) == mb_
btn_yes then
    echo('You chose "Yes"');
endif;
```

displays a message box with **Yes** and **No** buttons and a **Question** icon.

## *midlist*

```
midlist(lst, start, len)
```

Returns the section of list *lst* starting at position *start* of length *len*. *Start* counts from 0.

## Example

```
midlist('zero one two three four five', 2, 3)
```

Returns two three four

## *midstr*

```
midstr(str, start, len)
```

Returns the section of string *str* starting at position *start* of length *len*. *Start* counts from 0.

## Example

```
midstr('012345', 2, 3)
```

Returns 234

## *multisubst*

`multisubst(string, search-list, replace-list):`

Returns the string resulting from substituting all occurrences in *string* of each item in *search-list* by the corresponding item in *replace-list* (cf. `subst()`). *Search-list* and *replace-list* are a list of strings; if there are fewer items in *replace-list* than in *search-list*, occurrences of the extra items from *search-list* found in *string* are removed.

### Example

```
multisubst(var(text), split("& < > ^\"", ' '), "&amp; &lt; &gt; &quot;")
=> turn arbitrary text string into acceptable HTML string
multisubst("abc^rdef^r^n", "^r^n ^r", "^n")
=> "abcdef^n"
```

## *num\_records*

`num_records(database)`

Returns the total number of records in *database* (without scanning it). *Database* may be one of: `part, version, baseline, baseline_detail, issue, instance, monitor, cr, items_affected, status_log, vote, pool, class, workspace, role, cycle, command, field_name, rolemap, config, include, report_format, username, group`.

### Example

```
num_records('cr')
```

Returns the total number of CRs in the database

## *num\_records\_in\_range*

`num_records_in_range(database, index, item)`

Returns the number of records in *database*, indexed by *index*, which match *item*, without scanning the records. This is *much* faster than executing an `eachloop` which counts. *Database* identifies a "user-friendly" database: the possible values are as those to an `each/report`, *not* those acceptable to `num_records()`, viz.: *Index* may be used to identify which index to look at (like the `-indexedby` argument to `each/report`; if `''` is specified the default index is used. *Item* is an item specification like that to `each`; often `start-->end` will be used (which is a range), but a single item ending in a `*` (asterisk) (meaning count items which begin with the string), a plain single item (meaning count items with just that in the index), `any` and `none` are all supported. In other words, arguments are just like those to `each`.

### Examples

```
num_records_in_range('cr', '', 'RFC00000-->RFC99999')
how many (non-deleted) RFCs are there?
```

```
num_records_in_range('issue', '', '/sub1/comp1;*')
how many check-outs (anywhere for any purpose) of /sub1/comp1 are there?
```

```
num_records_in_range('baseline_detail', '', bl_name)
how many details are there in the current baseline?
```

```
num_records_in_range('cr', 'assignee', user)
how many CRs are currently assigned to me?
```

```
num_records_in_range('cr', 'assignee', 'none')
```

how many CRs are currently not assigned to anybody?

### *num\_votes*

`num_votes(database, item, status)`

Returns the number of sets of voting that have occurred in the *database* for the *item* for the *status*; effectively this is the number of "Vote Start" records. *database* should be `cr`, `part` or `baseline`.

### Example

```
num_votes('cr', 'RFC00010', 'CAB')
```

Might return 2 if there have been 2 rounds of voting in the CAB status for RFC00010

### *option*

`option(name)`

Returns the value of the option named *name* in the `config.ac` file; returns `' '` if no such entry is found. For convenience, if the corresponding value is `False` this function returns `' '` (false) so that it may be used as a truth value.

### Example

```
if option("JointEdit") then ...
if is_in_list(var(suffix), option("BinarySuffixes")) then ...
```

### *outstanding\_voter\_details*

`outstanding_voter_details(database, item, status)`

Returns a list of outstanding voters for *item* in *status* in *database*.

Each item in the list consists of a tab-separated string containing the voter, the voter type, and whether the voter is mandatory. The voter will be a user name, role name, group name or field name. The voter type specifies which of these types the voter is.

### Example

```
outstanding_voter_details('cr', 'RFC00010', 'CAB')
```

Might produce a list something like:

CAB	Field	Yes
change_manager	Role	No

If members of the CAB field and the change\_manager role are specified as voters with the CAB field users are mandatory voters and the change\_manager is optional.

### *pagenumber*

`pagenumber()`

Returns a string representing the current page number of a report. This function should only be used in report formats.

### *part\_browser\_reread*

```
part_browser_reread():
```

Causes the part browser (and viewer) windows to be re-read when the current command has completed. Similar to `file_browser_reread()`.

### *part\_component*

`part_component(part)`

Returns the component (i.e. any version removed) section of *part*. This function just treats *part* as a string: it does not access the database or require that *part* actually exist.

#### **Example**

```
part_component('/subsys/component;branch.004')
```

Returns `/subsys/component`

### *part\_filename*

`part_filename(part)`

Returns the filename related to the part *part*. The filename may be an actual file or a directory, depending on the type of the part. Note that this is name of the VC file for the part. The part must be a full pathname.

#### **Example**

```
part_filename('/subsys/component')
```

Might return `d:\acprojects\project1\VCFILES\subsys\VC\component`

### *part\_instance\_number*

`part_instance_number(instance)`

Return the instance number from *instance*.

#### **Example**

```
part_instance_number('/Car/CD Player;004:0023')
```

Returns `0023`

### *part\_name*

`part_name(part)`

Returns the name (including version, if any, but excluding instance, if any) section of *part*. This function just treats *part* as a string: it does not access the database or require that *part* actually exist.

#### **Example**

```
part_name('/subsys/component;branch.004')
```

Returns `component;branch.004`

```
part_name('/Car/CD Player;004:0023')
```

Returns `CD Player;004`

### *part\_name\_instance*

`part_name_instance(instance)`

Returns the name part of a full part path complete with the version and instance identifier.

**Example**

```
part_name_instance("/Hardware/PCBs/PCB1;003:00123)
```

returns PCB1;003:00123

***part\_parent***

```
part_parent(part)
```

Returns the parent section of *part*. This function just treats *part* as a string: it does not access the database or require that *part* actually exist.

**Example**

```
part_parent('/subsys/component;branch.004')
```

Returns /subsys

***part\_pool\_filename***

```
part_pool_filename(part, pool)
```

Returns the work filename associated with *part* for *pool*. The user must be attached to a workspace for this to work.

**Example**

```
part_pool_filename('/subsys/component', 'pool1')
```

Might return \\server\acpools\project\pool1\subsys\component

***partsaffecting***

```
partsaffecting(part)
```

Returns a list of the parts which affect *part*, if any. *Part* may refer to a subsystem, component or version. If *part* specifies a component name terminated by ";" then the list of parts will include all those which affect any version of *part*.

**Example**

```
partsaffecting('/product/doc/source.spec') ==> '/product/sw/source'
```

***part\_template\_text***

```
part_template_text(part-class)
```

Returns, as a list of lines, the text of the template file, if any, for new parts of class *part-class*.

**Example**

```
part_template_text('Source')
```

Returns the contents of the template file for parts of class 'Source', this might be for example:

Description of Part::

### *part\_text\_filename*

`part_text_filename(partname)`

Returns the filename for the part text corresponding to part *partname*. In general this is the same as the part name but with any punctuation characters removed. This is the name of the file that will be created by the `getpatext` command and used by the `putpatext` command.

#### **Example**

`part_text_filename('/SWProduct/Source/module1.c')` will return 'SWProductSourcemodule1c'

### *part\_text\_section*

`part_text_section(part-text, section)`

Returns a list of the lines in the specified *section* in *part-text*. Note that *part-text* is the actual part text (e.g. as produced by the field reference `pa_text` or `pa_comp_text` or by reading in a part text template), not a Part name.

#### **Example**

`part_text_section(pa_text, 'Description')`

Returns the contents of the Description section of the Part text for the current Part record

### *part\_text\_section\_list*

`part_text_section_list(part-text)`

Returns a list of all the section names in *part-text*. Note that *part-text* is the actual Part text (e.g. as produced by the field reference `pa_text` or `pa_comp_text` or by reading in a Part text template), not a Part name.

#### **Example**

`part_text_section_list(pa_text)`

Returns a list of the text sections names for the Part text for the current Part record, e.g.

Description  
Review Comments

### *part\_version*

`part_version(part)`

Returns the version (i.e. after the `;`, if any and before the `:`, if any) section of *part*. This function just treats *part* as a string: it does not access the database or require that *part* actually exist.

### Example

```
part_version('/subsys/component;branch.004')
```

Returns `branch.004`

```
part_version('/Car/CD Player;004:0023')
```

Returns `004`

### *part\_version\_branch*

```
part_version_branch(version)
```

Returns the branch (i.e. after the `;` and before the last `.`, if any) section of *version*; returns `'` if the version is not on a branch. This function just treats *version* as a string: it does not access the database or require that *version* actually exist.

### Example

```
part_version_branch('/subsys/component;branch.004')
```

Returns `branch`

### *part\_version\_instance*

```
part_version_instance(instance)
```

Return the version and instance number from *instance*.

### Example

```
part_version_instance('/Car/CD Player;004:0023')
```

Returns `004:0023`

### *part\_version\_number*

```
part_version_number(part)
```

Returns the number (i.e. after the `;` and after the last `.` (before a `:`), if any) section of *part*. This function just treats *part* as a string: it does not access the database or require that *part* actually exist.

### Example

```
part_version_number('/subsys/component;branch.004')
```

Returns `004`

```
part_version('/Car/CD Player;Branch.004:0023')
```

Returns `004`

### *part\_work\_filename*

```
part_work_filename(part)
```

Returns the work filename associated with *part* in the current workspace . The part must be a full path name to the part. The user must be attached to a workspace for this to work.

#### Example

```
part_work_filename('/subsys/component')
```

Might return `c:\workspace1\subsys\component`

#### *part\_workspace\_filename*

```
part_workspace_filename(part, workspace)
```

Returns the work filename associated with *part* for *workspace*. The part must be a full path name to the part.

#### Example

```
part_workspace_filename('/subsys/component', 'workspace1')
```

Might return `c:\workspace1\subsys\component`

#### *partsaffected*

```
partsaffected(part)
```

Returns a list of the parts which *part* affects, if any. *Part* may refer to a subsystem, component or version. If *part* specifies a component name terminated by ";" then the list of parts will include all those which any version of *part* affects.

#### Example

```
partsaffected('/product/sw/source') ==> '/product/doc/source.spec'
```

#### *paste\_clipboard*

```
paste_clipboard()
```

Pastes from the clipboard to the current edit control.

#### *paste\_string*

```
paste_string(str)
```

Pastes the specified string at the caret position in the current up-front window in ACE. This might be used in a user-defined function called from a toolbar button; for instance, the user could press a toolbar button to paste the current date into the CR text when in the CR Viewer.

#### *plain\_filename*

```
plain_filename(path)
```

Returns the plain filename from the operating system path *path*. The plain filename is the filename excluding the path.

#### Example

```
plain_filename('c:\dir\file.txt')
```

Returns `file.txt`

### *pos\_in\_list*

```
pos_in_list(str, list)
```

Returns the position (starting from 0) of string *str* in list *list*, or -1 if not present.

#### **Example**

```
pos_in_list('two', 'zero one two three')
```

Returns 2

```
pos_in_list('four', 'zero one two three')
```

Returns -1

### *project\_users*

```
project_users()
```

Returns a list of users valid for the current **AllChange** project. If the project is available to everyone, then this will be the same as [licensed\\_users](#). If it is available to a limited set of users in the [project definition](#) then it will return those users (Include named list). If it is limited to all licensed users except those specifically named in the project definition then it will return `licensed_users` less those excluded (Exclude named list)

#### **Example**

```
project_users()
```

Might return `fred jim sheila` if these are the licensed users and all users are permitted access to the project

### *prompt*

```
prompt(prompt-text, fillin, buttons)
```

Prompts for the user to input some information and returns his answer. This function may also be used just to present some information and wait for the user to acknowledge it.

In ACC the *prompt-text* will be shown and any further keyboard input until a RETURN is entered will be taken as the answer. In ACE a suitable dialog box will appear containing the *prompt-text* and appropriate fill-in and/or buttons.

*Prompt-text* is the text that will be displayed to the user. Multi-line text is permitted by embedding the new-line characters required by the operating system in the text (see [ACCEL Conditions](#)), e.g. "`^r^n`" under Windows.

*Fillin* specifies whether the prompt should present the user with a fill-in field (aka. edit control), and whether it should be a multi-line fill-in field. Specifying *false*, or the empty string, causes no fill-in to be displayed. In order to show a single-line fill-in, '*single*' should be specified, and to show a multi-line fill-in, '*multi*' or *true* should be specified.

*Buttons* is a (literal) string indicating what buttons, if any, to offer the user; it must be one of the following:

<code>Ok</code> or <code>' '</code>	"OK" button
<code>OkCancel</code>	"OK" and "Cancel" buttons
<code>YesNo</code>	"Yes" and "No" buttons
<code>YesNoCancel</code>	"Yes", "No" and "Cancel" buttons

If used from ACC the function informs the user what "buttons" are available, offers a default, and requires that the user types the button text for a non-default selection. Note that the 'fill-in', if specified, is always inherently a single-line, as pressing the return key will return the typed text immediately.

The function returns a string: `cancel` if there is a "Cancel" button and the user selects it, otherwise whatever the user enters if there is a fill-in or the text of the button selected (`ok`, `yes`, `no`, etc) if there is not.

There is an HTML based version of this function which will be used when the function is invoked from the browser interface. The HTML version of the function is called `web_prompt` and is defined in the ACCEL function file `webifunc.ac`. Note that if a fill-in is specified, it is always a single-line edit control.

## Example

```
prompt("Enter something", 'multi', '')
prompt("This is a message^r^nAcross two lines", '', '')
setvar(ans, prompt("Create a branch?", '', 'YesNoCancel'))
```

## *prompt\_blinelist*

```
prompt_blinelist(index, indexvalue, condition, showlast, extra-list, select-type)
```

Displays the baseline selection dialog. *Index* and *indexvalue* specify the name (as per [display index names](#)) and value of the field to filter on, if any. *Condition* specifies any condition to filter on. *show-last* specifies the number of baseline versions to be shown. This should be a number greater than 0 or (All). *Extra-list* specifies a list of any special extra values to include in the list (e.g. (Default)). *Select-type* should be set to one of `single`, `multi` or `none` to allow single, multiple or no (use for information only purpose) item selection. Returns the names of the baselines selected (or `cancel`).

If no *index*, *indexvalue* or *condition* is specified (i.e. the empty string is passed) then the last saved settings from the baseline browser will be used. If 'None' is specified for a parameter, then the dialog's parameter will not be set.

There is an HTML based version of this function which will be used when the function is invoked from the browser interface. The HTML version of the function is called `web_prompt_blinelist` and is defined in the ACCELfunction file `webifunc.ac`

## Example

```
setvar(bline, prompt_blinelist('', '', '', 0, '', 'single'))
```

## *prompt\_blineversionlist*

```
prompt_blineversionlist(basename, show-last, extra-list, select-type)
```

Prompts the user to select a baseline version corresponding to *basename*. *show-last* indicates the number of versions to show. This should be a number greater than 0 or (All). *Extra-list* specifies a list of any special extra values to include in the list (e.g. (Default)). *Select-type* should be set to one of `single`, `multi` or `none` to allow single, multiple or no (use for information only purpose) item selection. Returns the version selected (or `cancel`).

There is an HTML based version of this function which will be used when the function is invoked from the browser interface. The HTML version of the function is called `web_prompt_blineversionlist` and is defined in the ACCELfunction file `webifunc.ac`

## Example

```
setvar(version, prompt_blineversionlist('bline', '(All)', '', 'single'))
```

### *prompt\_command*

```
prompt_command(command-dialog, items)
```

Displays the *command-dialog* specified, with *items* in the main item(s) fill-in (e.g. **Parts** for the **Alter Part** command). *Command-dialog* can be any of the dialogs valid for the **ShowDialog** action in **Menu Items** (see [Menu Actions](#)). Returns the command line necessary to execute the command with the selected options (or cancel).

#### **Example**

```
setvar(Command, prompt_command('Copy', var(part)))
```

### *prompt\_command\_values*

```
prompt_command_values(command-dialog, items, value-list)
```

Essentially the same as [prompt\\_command](#), but with an additional parameter to allow initial values to be specified. These values are specified as a list of <field>=<value> pairs. Where the dialog field corresponds directly to a user-defined field or a built-in database field, the name may be specified as either the back-end field name or the display name. For fields that exist only on the dialog, the field should be specified as the label for the dialog's field.

#### **Example**

```
setvar(pcommand, prompt_command_values('Add_CR', '', "Class=Problem"));
if var(pcommand) != 'Cancel' then
interpret(var(pcommand));
endif;
```

Prompts the user with the Add CR dialog with the Class initialised to Problem. The command line to execute the command is returned from the `prompt_command_values` function and then executed using the [interpret](#) function.

### *prompt\_condedit*

```
prompt_condedit(context)
```

Prompts the user to enter a condition into the ACCELCondition Editor. The fields available are those appropriate to *context*, which may be any of the contexts defined in `condedit.ac` (e.g. CR or Baseline)—see [ACCEL Condition Editor](#). Returns the condition entered (or cancel).

There is an HTML based version of this function which will be used when the function is invoked from the browser interface. The HTML version of the function is called `web_prompt_condedit` and is defined in the ACCEL function file `webifunc.ac`

#### **Example**

```
prompt_condedit('CR')
```

### *prompt\_crlist*

```
prompt_crlist(index, indexvalue, condition, extra-list, select-type)
```

Displays the CR selection dialog. *Index* and *indexvalue* specify the name (as per [display\\_index\\_names](#)) and value of the field to filter on, if any. *Condition* specifies any condition to filter on. *Extra-list* specifies a list of any special extra values to include in the list (e.g. (Default)). *Select-type* should be set to one of `single`, `multi` or `none` to allow single, multiple or no (use for information only purpose) item selection. Returns the numbers of the CRs selected (or `cancel`).

If no *index*, *indexvalue* or *condition* is specified (i.e. the empty string is passed) then the last saved settings from the CR browser will be used. If 'None' is specified for a parameter, then the dialog's parameter will not be set.

There is an HTML based version of this function which will be used when the function is invoked from the browser interface. The HTML version of the function is called `web_prompt_crlist` and is defined in the ACCELfunction file `webifunc.ac`

### Example

```
setvar(crs, prompt_crlist('', '', "cr_class == 'Problem'", '', 'multi'))
```

### *prompt\_dirlist*

```
prompt_dirlist(title, initial-directory)
```

Displays a common directory selection dialog and allows selection of a directory. *Title* specifies the dialog title and *initial-directory* the directory to start in (if desired). Returns the directory selected (or `cancel`).

There is an HTML based version of this function which will be used when the function is invoked from the browser interface. The HTML version of the function is called `web_prompt_dirlist` and is defined in the ACCELfunction file `webifunc.ac`

### Example

```
prompt_dirlist('Select include directory', cwd)
```

### *prompt\_editfile*

```
prompt_editfile(title, filename)
```

Causes a dialog to be displayed containing the contents of the specified file in a multi-line edit control. The text may be edited and saved by the user. The function returns whether or not the text was changed.

### Example

```
prompt_editfile('Here is the file to edit', 'c:\autoexec.bat')
```

### *prompt\_filelist*

```
prompt_filelist(title, initial-directory, initial-file, patterns, type, select-type)
```

Displays the common file selection dialog and allows selection of files. (As the file selection dialog does not allow the selection of a directory, `prompt_dirlist()` should be used instead when a directory is required.) Returns a list of the file(s) selected (or `cancel`).

*Title* specifies the dialog title. *Initial-directory* is the directory to start in (current directory if empty). *Initial-file* is an initial file selection, if desired. *Patterns* should specify any patterns as descriptive and pattern pairs, separated by | (vertical bar) characters, e.g.:

```
ACCEL (*.ac)|*.ac|Source (*.c;*.h)|*.c;*.h
```

if *patterns* is empty all files are shown. *Type* specifies the behaviour of dialog: `open` allows only existing files to be selected, `save` warns if an existing file is selected, `any` allows any file to be selected. *Select-type* should be set to either `single` or `multi` to allow single or multiple item selection.

There is an HTML based version of this function which will be used when the function is invoked from the browser interface. The HTML version of the function is called `web_prompt_filelist` and is defined in the ACCEL function file `webifunc.ac`

### Example

```
setvar(files, prompt_filelist('Select Files to Attach', '', '', '', 'open',
'multi'))
```

### *prompt\_formatlist*

```
prompt_formatlist(pattern)
```

Prompts the user to select a report format file matching the specified (operating system) pattern. If *pattern* is empty then a pattern of `*.rep` is used. Returns the file selected (or `cancel`).

There is an HTML based version of this function which will be used when the function is invoked from the browser interface. The HTML version of the function is called `web_prompt_formatlist` and is defined in the ACCEL function file `webifunc.ac`

### Example

```
setvar(mswordrepfile, prompt_formatlist("mw*"))
```

### *prompt\_instancelist*

```
prompt_instancelist(prompt, subsys, vers, filter, condition, select-type)
```

Prompts the user to enter an instance(s) or select from a part list displaying the contents of the specified subsystem and matching the specified condition. Returns a list of the instance(s) selected (or `cancel`).

*Prompt* specifies the text to appear in the initial prompt dialog; if it is `None` this dialog is skipped and the part list is displayed immediately.

If *subsys* is empty the current subsystem is displayed.

*Vers* indicates whether components *or* versions should be displayed and, if versions, which version(s): it should be one of `Component`, `All`, `All Edit`, `Default`, `Registered`, `Top`.

*Filter* indicates which components/ versions to display: it should be one of the values shown in the part browser **Filter** drop down list or `'` to disallow components and versions completely from selection (i.e. only subsystems may be selected). Note that the part browser filter list will vary according to any nomenclature mapping in effect for the Issue term, out-of-the box this is mapped to `check-out`.

If *condition* is not empty then only parts satisfying the condition are displayed, e.g. `match_wild(part, '*.c')`.

*Select-type* should be set to either `single` or `multi` to allow single or multiple item selection.

There is an HTML based version of this function which will be used when the function is invoked from the browser interface. The HTML version of the function is called `web_prompt_instancelist` and is defined in the ACCELfunction file `webifunc.ac`

### Example

```
setvar(instances, prompt_instancelist('None', "", 'Component', 'All', '',
'multi'))
```

### *prompt\_list*

```
prompt_list(prompt, list)
```

Prompts the user to choose one or more items from a scrolling list displaying items given in *list*. Any items in *list* which are prefixed with a '+' will be pre-selected. Returns a list of the item(s) selected (or `cancel`).

There is an HTML based version of this function which will be used when the function is invoked from the browser interface. The HTML version of the function is called `web_prompt_list` and is defined in the ACCELfunction file `webifunc.ac`

### Example

```
setvar(users, prompt_list("Select Users", sortlist(user_to_full-
namelist(var(users)), '')))
```

### *prompt\_list\_columns*

```
prompt_list_columns(prompt, list, headings)
```

Prompts the user, with text in *prompt*, to choose one or more items from a scrolling list displaying items given in *list*. Any items in *list* which are prefixed with a '+' will be pre-selected. *Headings* may specify a list of column headings/titles in which case the items in *list* should contain column text for each column separated by tab characters.

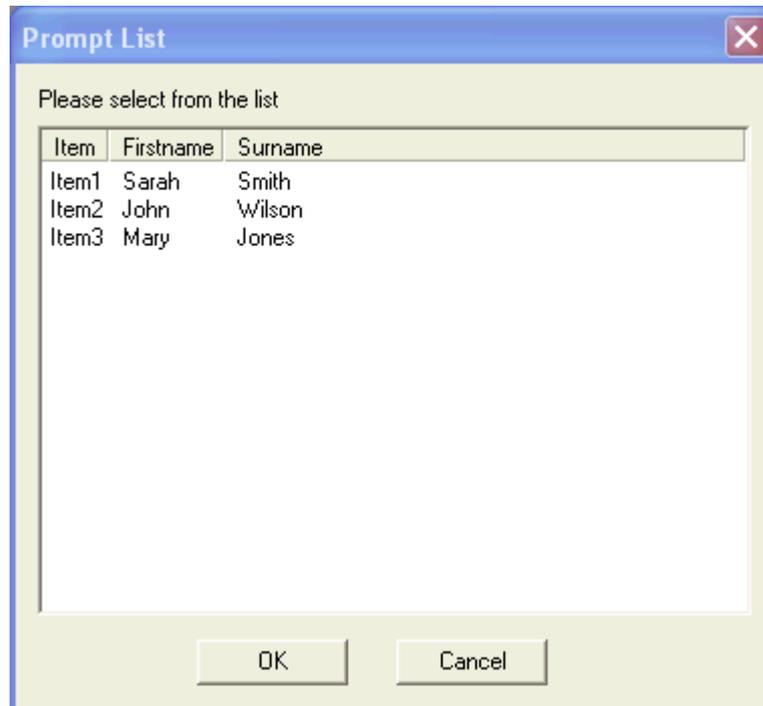
All columns are automatically sized so that all item text and heading text is visible

Returns a list of the item(s) selected (or `cancel`).

There is an HTML based version of this function which will be used when the function is invoked from the browser interface. The HTML version of the function is called `web_prompt_list` and is defined in the ACCELfunction file `webifunc.ac`

### Example

```
prompt_list_columns
("Please select from the list",
split("Item1^tSarah^tSmith|Item2^tJohn^tWilson|Item3^tMary^tJones", '|'),
"Item Firstname Surname")
```



### *prompt\_partlist*

`prompt_partlist(prompt, subsys, allowsubs, vers, filter, condition, select-type)`

Prompts the user to enter a part(s) or select from a part list displaying the contents of the specified subsystem and matching the specified condition. Returns a list of the part(s) selected (or `cancel`).

*Prompt* specifies the text to appear in the initial prompt dialog; if it is `None` this dialog is skipped and the part list is displayed immediately.

If *subsys* is empty the current subsystem is displayed.

*Allowsubs* is a truth indicating whether or not subsystems should be able to be returned (*allowsubs* true). Subsystems may only be returned by selecting and clicking **OK**, as double-clicking on a subsystem navigates down into the subsystem.

*Vers* indicates whether components *or* versions should be displayed and, if versions, which version(s): it should be one of `Component`, `All`, `All Edit`, `Default`, `Registered`, `Top`, or `' '` to disallow versions completely.

*Filter* indicates which components/ versions to display: it should be one of the values shown in the part browser **Filter** drop down list or `' '` to disallow components and versions completely from selection (i.e. only subsystems may be selected). Note that the part browser filter list will vary according to any nomenclature mapping in effect for the Issue term, out-of-the box this is mapped to check-out.

If *condition* is not empty then only parts satisfying the condition are displayed, e.g. `match_wild(part, '*.c')`, if *condition* is empty then the last saved settings from the parts browser is used, if `'None'` is specified then the *condition* is not set.

*Select-type* should be set to either `single` or `multi` to allow single or multiple item selection.

There is an HTML based version of this function which will be used when the function is invoked from the browser interface. The HTML version of the function is called `web_prompt_partlist` and is defined in the ACCELfunction file `webifunc.ac`

**Example**

```
setvar(parts, prompt_partlist('None', "", false, 'Component', 'All', '',
'multi'))
```

***prompt\_subsyslist***

```
prompt_subsyslist(title, initial-subsystem)
```

Displays a common subsystem selection dialog and allows selection of a subsystem. *Title* specifies the dialog title and *initial-subsystem* the directory to start in (if desired). Returns the subsystem selected (or cancel).

There is an HTML based version of this function which will be used when the function is invoked from the browser interface. The HTML version of the function is called `web_prompt_subsyslist` and is defined in the ACCELfunction file `webifunc.ac`

**Example**

```
prompt_subsyslist('Pick a subsystem', '/')
```

***prompt\_userlist***

```
prompt_userlist(prompt, show-groups, selection, selected-items, extra-items, exclude-items)
```

Prompts the user to choose one or more user-names from a scrolling list of licensed users. The function returns a list of selected items, as back-end usernames, as a string, as per [prompt list](#). The *prompt* is shown at the top of the dialog. If *show-groups* is true then the list will include all defined group names, which will be shown in bold. The *selection* parameter may be 'single' or 'multi' for single- or multi-selection. *Selected-items* specifies a list of items to be pre-selected in the list. *Extra-items* allows additional items to be added to the list (e.g. "(None)"). The *exclude-items* parameter specifies a list of items to exclude from the generated user list.

There is an HTML based version of this function which will be used when the function is invoked from the browser interface. The HTML version of the function is called `web_prompt_list` and is defined in the ACCELfunction file `webifunc.ac`.

**Example**

```
setvar(users, prompt_userlist("Select initial assignee", true, 'single',
user, '(None)', role_users('/subsys', 'developer')))
```

Displays a prompt list showing licensed users and all defined groups, but excluding users who have the role developer for part /subsys. The list is single-select, and the current user's name is initially selected.

***prompt\_versionlist***

```
prompt_versionlist(component, what-versions, extra-list, select-type)
```

Prompts the user to select (a) version(s) from a version list displaying the versions of the specified component. Returns a list of the version(s) selected (or cancel).

*Component* specifies the component whose versions are displayed. *Extra-list* specifies a list of any special extra values to include in the list (e.g. (Default)). *What-versions* determines which versions are to be included in the list as per the `version_list()` function. *Select-type* should be set to one of `single`,

`multi` or `none` to allow single, multiple or no (use for information only purpose) item selection. Returns the version(s) selected (or `cancel`).

There is an HTML based version of this function which will be used when the function is invoked from the browser interface. The HTML version of the function is called `web_prompt_versionlist` and is defined in the ACCELfunction file `webifunc.ac`

### Example

```
setvar(version, prompt_versionlist(var(part), 'all', '', 'single'))
```

### *protfile*

```
protfile(filename)
```

Protects the file *filename* (i.e. makes the file read only).

### Example

```
protfile('c:\workspace\file')
```

Makes `c:\workspace\file` read-only

### *protserverfile*

```
protserverfile(filename)
```

If running client/server gets the server to protect (make read-only) a file; *filename* should specify a path from the server's point of view. If not running client/server acts the same as `protfile()`.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

### Example

```
protserverfile(join_server_paths(acserverprojdir, 'file'))
```

Gets the server to make file in the project directory read-only

### *putaway*

```
putaway(args, part, keep)
```

Calls the VC **Putaway** tool with arguments *args* on the VC file corresponding to *part*. If *keep* is true the workfile is retained after the putaway, otherwise it is deleted. The *keep* option is used to indicate that the keep option (`-k`) has been specified on the command line

It is most important that this mechanism should be used when calling **Putaway** on a VC file which has the **Putaway when no differences** flag set to not create a new version when there are no differences since the `putaway` function will detect this and remove the version from the **AllChange** database when no new version is created.

For information on valid arguments see the **AllChange VC Tools** manual.

This function *must* be used instead of `command`.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

### *putserverfile*

`putserverfile(fromfile, tofile)`

If running client/server puts (copies) a file from the client to the server; *fromfile* should specify a path from the client's point of view while *tofile* should specify a path from the server's point of view. The file's permissions and timestamp are retained. If not running client/server acts the same as `copyfile()`.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

#### **Example**

```
putserverfile('c:\temp\test1.ac', join_server_paths(acserverprojdir,
'test1.ac'))
```

Copies the file test1.ac from the client's c:\temp directory to the server's project directory

### *quote*

`quote(str)`

Returns *str* with whatever quoting is necessary (if any) to make it acceptable as a single partname or filename for the **AllChange** interpreter or operating system commands.

#### **Examples**

```
quote('filename') == 'filename' (no quoting)
quote('file name') == '"file name"' (quote space)
quote('A Part Name') == '"A Part Name"' (quote space)
```

### *quote\_accel*

`quote_accel(str)`

Returns *str* with whatever quoting is necessary (if any) to make it acceptable as an ACCELstring.

#### **Example**

```
quote_accel('abc ^" def') == '"abc ^^^" def"'
```

### *quote\_accellist*

`quote_accellist(list)`

Returns *list* with whatever quoting is necessary (if any) to make each element acceptable as an ACCEL string.

#### **Example**

```
quote_accellist('abc ^" def')
```

Returns "abc" "^^^" def"

### *quote\_arg*

`quote_arg(str)`

Returns *str* with whatever quoting is necessary (if any) to make it acceptable as a single argument for the **AllChange** interpreter.

#### **Example**

```
interpret('return -edit -comment '.quote_arg(getarg('-other'))...)
```

### *quote\_arglist*

`quote_arglist(list)`

Returns *list* with whatever quoting is necessary (if any) to make each element acceptable as a single argument for the **AllChange** interpreter.

### *quote\_os*

`quote_os(str)`

Returns *str* with whatever quoting is necessary (if any) to make it acceptable as a single word for the operating system command line.

#### **Example**

```
quote_os('filename') == 'filename' (no quoting)
quote_os('file name') == '"file name"' (quote space)
quote_os(' "\$*') == '" \\\$*"' (MSOPSYS)
```

### *quote\_oslist*

`quote_oslist(list)`

Returns *list* with whatever quoting is necessary (if any) to make each element acceptable as a single word for the operating system command line.

### *quotelist*

`quotelist(list)`

Returns *list* with whatever quoting is necessary (if any) to make each element of the list acceptable as a single partname or filename for the **AllChange** interpreter or operating system commands.

### *read\_branches*

`read_branches()`

This function reads the branches definition file into ACE/ACC.

#### **Examples**

`read_branches()` this will cause the branches definition file to be re-read.

### *read\_functions*

`read_functions(filename, readonly)`

Allows a file just like the `cmdfunc.ac` system file to be read into an already running **AllChange** session. The file may contain user-defined functions. *Filename* should name the file; a relative filename is sought first in the **AllChange** project directory (i.e. the same directory as `commands.ac`), then in the system directory and finally in the user's home directory. If *readonly* is true the functions in the file may not normally be redefined by a subsequent `read_functions()`, if it is false they may be redefined; however, *readonly* is ignored if the file is found in the user's home directory, or if the user is an administrative user: any functions read from the file may always be redefined. (Note that all command definitions read from the `commands.ac` file are automatically marked read-only and so may not be redefined for safety.)

This function is especially useful while setting up a complex system: develop functions in a file and keep going `read_functions('other', '')` from within **AllChange** (from ACETHIS could be done from the user-defined functions menu for speed). This allows syntax errors etc. to be recognised quickly without having to exit and reenter **AllChange**. When complete, however, it is suggested that the file be included into `includes.ac`.

### Example

```
read_functions('projfunc.ac', '')
```

Rereads `projfunc.ac`

### *read\_lockouts*

```
read_lockouts()
```

Re-reads the lockout definitions for the current **AllChange** project. This will re-read the `lockout.ac` and file.

### *read\_pools*

```
read_pools()
```

Re-reads the pool definitions for the current **AllChange** project. This will re-read the `pools.ac` and `ftppool.ac` files.

### *read\_report\_summaries*

```
read_report_summaries()
```

Re-reads the `report.sum` file(s) — see [Report Formats](#).

### *read\_rolemap*

```
read_rolemap()
```

Re-reads the command access definitions for the current **AllChange** project. This will re-read the `rolemap.ac` file.

### *read\_roles*

```
read_roles()
```

Re-reads the role definitions for the current **AllChange** project. This will re-read the `roles.ac` file.

### *read\_workspaces*

```
read_workspaces()
```

Re-reads the workspace definitions for the current **AllChange** project. This will re-read the `workspcs.ac` and `ftpwspc.ac` files.

The user will be detached from the current workspace, and re-attached after the re-reading is completed. If the last workspace is no longer valid for the user after re-reading, then the user is left unattached to a workspace.

### *reg\_key\_value\_str*

`reg_key_value_str(key, valuname)`

Returns the string value for the value name *valuname* associated with registry key *key*. *Key* specifies the full path of the registry key. *Valuname* specifies the name of the value desired; if it is '' (empty string) the key's unnamed/ default value is returned. If the value's type is not a string it is converted to a string representation. If the key/ value exists but its value is empty or not defined (`empty`) is returned. If (and only if) the key/ value does not exist '' is returned.

#### **Example**

```
reg_key_value_str('HKEY_CLASSES_ROOT\.txt', 'Content Type')
```

Returns `text/plain`

### *relative\_part*

`relative_part(part1, part2)`

Returns that section of part *part2* which is relative to *part1*, or '' if *part2* is not relative to *part1*.

#### **Example**

```
relative_part('/subsys1', '/subsys1/subsys2/component')
```

Returns `subsys2/component`

### *relative\_path*

`relative_path(path1, path2)`

Returns that section of operating system path *path2* which is relative to *path1*, or '' if *path2* is not relative to *path1*.

#### **Example**

```
relative_path('c:\Temp', 'c:\Temp\dir\file')
```

Returns `dir\file`

### *renamecr*

`renamecr(oldcr, newcr)`

Renames CR *oldcr* to *newcr*. It renames the associated CR text and parts affected etc. It is intended as an aid for converting existing CR numbers to new style.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

### *renameversion*

```
renameversion(oldversion, newversion)
```

Renames *oldversion* to *newversion*. *Oldversion* should specify the full path of the version (e.g. `/subsys/component;branch.001`); *newversion* should specify only the desired new version section (e.g. `branch.001`). It renames any associated CRs, versions solved, baseline details etc. It is intended as an aid for converting old non-fixed-digit version numbers to new fixed-digit style. Use with care.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

### *renfile*

```
renfile(filename, newfilename)
```

Renames the file *filename* to *newfilename*. *Newfilename* should include the path to the new file if it is not to be in the current directory; it may cause the file to be moved to a different directory if desired. This function renames a file even if it is read-only; it also deletes any existing destination file before renaming (even if read-only). A fatal error is raised if the rename fails.

#### **Example**

```
renfile('oldfile', 'newfile')
```

Renames oldfile to newfile in the current directory

```
renfile('c:\Temp\temp001', join_paths(cwd, 'keepfile'))
```

Moves c:\Temp\temp001 to the current working directory, and renames it to keepfile

### *renserverfile*

```
renserverfile(fromfile, tofile)
```

If running client/server gets the server to rename the file *fromfile* to the file *tofile*; *fromfile* and *tofile* should specify paths from the server's point of view. If not running client/server acts the same as `renfile()`.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

### *renvcfile*

```
renvcfile(vcfilename, newvcfilename)
```

Renames the VC file *vcfilename* to *newvcfilename*. *Newvcfilename* should include the path to the new file if it is not to be in the current directory; it may cause the file to be moved to a different directory if desired.

This function *must* be used instead of `renfile`.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

### *repcommand*

```
repcommand(command, outputfile)
```

Issues the operating system command *command*. The operating system command should create a file as named by *outputfile*. The content of this output file is then read and returned as a list of lines from the file as the result of the function. Finally the output file is deleted.

If the empty string ( ' ' ) is specified as the command the output file is expected to already exist and is simply read and returned as a list of lines; in this case it is not deleted. If the command is just a single space ( ' ' ) the same is true except the file is deleted.

This function is intended for use in reports to allow information from external systems to be included in an **AllChange** report: for example, detailed information about the versions held in a VC file may be requested from the via a `vcrep` command.

### Example

```
setvar(lines, repcommand('', var(tempfile)))
```

Reads the content of the file named by `var(tempfile)` as a list of lines

### *report\_format\_list*

```
report_format_list()
```

Returns a list of all the report formats named in any `report.sum` file.

### *rep\_str*

```
rep_str(str, num)
```

Returns a string which is made up of *num* repetitions of *str*. This function may be useful, for example, for outputting a number of spaces which varies according to some parameter (see the tree report for parts for an example of this).

### Example

```
rep_str('three ', 3)
```

Returns three three three

### *rinstr*

```
rinstr(str1, str2)
```

Returns the position of the last occurrence of *str2* in *str1*, cf. `instr()`.

### Example

```
rinstr('0123456789012', '1')
```

Returns 11

```
rinstr('0123456789012', 'Z')
```

Returns -1

### *role\_users*

```
role_users(part, role)
```

Returns a list of the users who may have the role *role* on the part *part*.

### Example

```
role_users('/subsys', 'developer')
```

Returns the users who are developers for the subsystem /subsys

### *seconds\_to\_date*

`seconds_to_date(seconds):`

Returns the date/time corresponding to a number of seconds elapsed since 1970/01/01 00:00:00 UTC (GMT). The date/time will be in standard Intasoft format (YYYY/MM/DD hh:mm:ss).

#### **Example**

```
Seconds_to_date(file_time('c:\autoexec.bat'))
```

Returns the date/time of the file c:\autoexec.bat

### *server\_set\_env\_var*

`server_set_env_var(env-var-assignment)`

Sets an environment variable on the server c/f [set\\_env\\_var](#). *Env-var-assignment* should be in the same format as what follows the `-E` command-line argument recognised by **AllChange** programs, i.e. *var=value*.

#### **Example**

```
set_env_var('INCLUDE=c:\')
```

Sets the environment variable INCLUDE on the server to the value c:\

### *server\_env\_var*

`server_env_var(env-var-name):`

If running client/server gets the server to return the value of the environment variable named *env-var-name*. If not running client/server acts the same as `env_var()`.

This function is classed as "dangerous".

#### **Example**

```
server_env_var('PATH')
```

Returns the server's PATH variable

### *server\_oscommand*

`server_oscommand(command, command-line, input-file, output-file):`

If running client/server gets the server to execute an operating system command. *command* names the command (i.e. program) to execute, including leading path if appropriate; *command-line* specifies any command-line parameters; *input-file* and *output-file* specify the name of a file to be used as the command's standard input/output respectively. If *input-file* is empty ("") the command has no standard input. If *output-file* is empty ("") then any output (together with standard error) are displayed by the client. The command and all paths must be acceptable to the server. If not running client/server acts similarly to `command()`.

This function is classed as "dangerous". Furthermore, for the sake of extra security, ACSERVER looks for a file named `acsoscmd.ac` in its **AllChanges** system directory(only). If this file exists then it should contain the names of commands (no path but including suffix), one per line, which are allowed to be executed in this fashion; ACSERVER will then refuse to execute any command not named in this file. If the file has no

entries in it at all (other than comments/blank lines) then *any* command may be executed; if the file does not exist then *no* commands may be executed.

This facility is not used by AllChange for any standard client/server operations. It is only needed for some integrations with third-party products.

### ***server\_temp\_filename***

`server_temp_filename()`:

If running client/server gets the server to create a unique temporary filename and return its name; this should be deleted with `delserverfile()`.

If not running client/server acts similarly to `temp_filename()` (but takes no parameters).

This function is classed as "dangerous".

### ***serverdir\_exists***

`serverdir_exists(dirname)`

If running client/server gets the server to return whether a directory exists; *dirname* should specify a path from the server's point of view. If not running client/server acts the same as `dir_exists()`.

#### **Example**

```
serverdir_exists(acserversysdir)
```

Returns `true`

### ***serverdir\_size***

`serverdir_size(directory, recursive)`

Returns the size of files contained in the specified directory on the server. If `recursive` is true then sizes of sub-directories are also included recursively.

#### **Example**

```
serverdir_size("C:\tmp", false)
```

Might return 1234 as the size of the files in the directory `C:\tmp` on the server but excluding any sub-directories.

### ***serverfile\_exists***

`serverfile_exists(filename)`

If running client/server gets the server to return whether a file exists; *filename* should specify a path from the server's point of view. If not running client/server acts the same as `file_exists()`.

#### **Example**

```
serverfile_exists('c:\autoexec.bat')
```

Returns whether `c:\autoexec.bat` exists on the server

### ***serverfile\_size***

`serverfile_size(filename)`

Returns the size of the specified server file

**Example**

```
serverfile_size("C:\AllChange\ACSYS\commands.ac")
```

Might return 84036 as the size of the `commands.ac` file in the **AllChange** system directory on the server.

***serverfile\_time***

```
serverfile_time(filename)
```

If running client/server, gets the server to return the last modification time of file "filename", as per [file\\_time\(\)](#) (i.e. as the number of seconds elapsed since 1970/01/01 00:00:00 UTC). The *filename* parameter should specify a path from the server's point of view. If not running client/server, this function acts the same as [file\\_time\(\)](#).

**Example**

```
serverfile_time('c:\autoexec.bat')
```

returns the last modification time of `c:\autoexec.bat` on the server

***serverfile\_writable***

```
serverfile_writable(filename):
```

If running client/server gets the server to return whether a file is writable (i.e. not read-only); filename should specify a path from the server's point of view. If not running client/server acts the same as `file_writable()`.

**Example**

```
serverfile_writable('c:\autoexec.bat')
```

Returns whether `c:\autoexec.bat` is writable on the server

***set\_app\_title***

```
set_app_title(maintitle, subtitle)
```

Sets the title of the (main) application window to:

*maintitle* - *subtitle*

***set\_cnum\_resource\_value***

```
set_cnum_resource_value(resnum, numvalue)
```

Sets the value of the CR numbering counter associated with *resnum* to the number *numvalue*. *Resnum* is a digit between 0 and 9, corresponding to the [counter](#) specified in the CR class definition. *Numvalue* is a number: the next CR generated using that counter will be numbered *numvalue*+1.

This function is classed as "dangerous" — see ["Dangerous" Functions](#). It should only be used if an Administrator has some reason to want to reset the numbering for a class of CRs.

The current value of a resource/counter may be obtained using [get\\_cnum\\_resource\\_value\(\)](#).

**Example**

```
set_cnum_resource_value(1, 1000)
```

Will reset the first CR numbering counter to 1000. The next CR created of a class using that counter will be numbered 01001. This is used by the SCR class in the [Intasoft Standard](#) out-of-the-box configuration so the next SCR generated will be numbered SCR01001.

### ***set\_env\_var***

```
set_env_var(env-var-assignment)
```

Sets an environment variable. *Env-var-assignment* should be in the same format as what follows the `-E` command-line argument recognised by **AllChange** programs, i.e. *var=value*.

#### **Example**

```
set_env_var('INCLUDE=c:\')
```

Sets the environment variable INCLUDE to the value `c:\`

### ***set\_file\_time***

```
set_file_time(filename, time)
```

Sets the last modification time of *filename* to *time*. *time* is specified as the number of seconds elapsed since 1970/01/01 00:00:00 UTC (as returned by `file_time()`); if *time* is 0 then the current time is set.

#### **Example**

```
set_file_time('c:\autoexec.bat', 0)
```

Sets the last modification time of `c:\autoexec.bat` to now

### ***set\_ini\_var***

```
set_ini_var(registry-subkey, var-name, value)
```

Writes the entry *var-name=value* into the specified *registry-subkey* in the subkey for the **AllChange** project.

If *intasoftini* files are in use instead of the registry then the *registry-subkey* is treated as the section of the `.ini` file.

### ***set\_putaway\_cmd\_result***

```
set_putaway_cmd_result(value)
```

Sets the value of the `putaway_cmd_result` ACCEL variable to the *value* specified. This should be used with caution when wishing to simulate the result of a putaway command, e.g. when the `no_file` flag is in use.

The value may be:

0 - a new version has been created

1 - an error occurred

2 - no new version has been created because there were no differences

#### **Example**

```
set_putaway_cmd_result(2)
```

Sets the `putaway_cmd_result` ACCEL variable to the value 2.

***setvar***

```
setvar(var-name, value)
```

Assigns *value* to the user-defined variable named *var-name*.

**Example**

```
setvar(variable, 'value')
```

***setvar\_list***

```
setvar_list(var-name-list, value-list)
```

Sets each user-defined variable named in *var-name-list* to the corresponding value in *value-list*. If there are more items in *value-list* than in *var-name-list* the excess are ignored; if there are more items in *var-name-list* than in *value-list* the excess variables are set to ' '. This function may be useful for accessing space-separated fields in a string (or a separator other than space in conjunction with `split()` and `split-quote()`).

**Example**

```
setvar_list('one two three', '1 2 3')
```

***shell\_execute***

```
shell_execute(verb, file, parameters)
```

Performs a `ShellExecute()` (similar to what happens in Explorer if the user double-clicks or right-clicks on a file). *Verb* may be any of the recognised verbs (e.g. `edit`, `open`); if it is ' ' the `open/` default verb is used. *File* specifies a file which has a file association in Explorer, or an executable program. *Parameters* specifies any parameters to be passed to the executable program. The function returns a truth indicating whether the shell successfully performed the execute.

**Example**

```
shell_execute("edit", 'file.c', '')
```

Calls up the user's preferred editor for C files on file.c

***shell\_renfile***

```
shell_renfile(filename, newfilename)
```

Performs a `SHFileOperation(FO_RENAME)` (similar to what happens in Explorer if the user tries to rename a file in place) to rename *filename* to *newfilename*, cf. `renfile()`. A fatal error is raised if the rename fails.

***shell\_show\_file***

```
shell_show_file(path)
```

Under Windows, returns whether the shell (Explorer) would display the specified file/ directory. It does this by seeing whether the file or directory is *hidden* and, if so, looking at Explorer's current setting as to whether to show hidden files (this requires Windows 98 or 2000, or Internet Explorer 4 or later to have been installed with the *integrated shell* under Windows 95 or NT 4); otherwise returns true.

### *show\_floating\_output*

```
show_floating_output(show)
```

Tells ACE whether to show (up-front) the "floating" Output Window (i.e. not constrained within the main ACE window) the next time it decides to show output. *Show* is a truth. This is intended to be used by functions when ACE is running minimised, responding to requests from external programs, e.g. Explorer, and so the normal Output Window would not be visible.

### *show\_floating\_progress*

```
show_floating_progress(show)
```

Tells ACE whether to show (bring up-front) a *floating* progress window (i.e. not constrained within the main ace window); this window displays the contents of the status and progress bars. *show* is a truth. It is important to hide the window when the operation is completed. This is intended to be used by functions when ace is running minimised, responding to requests from external programs, e.g. Explorer, and so the normal status and progress bars would not be visible.

### *show\_output*

```
show_output(show)
```

Tells ace whether to show (up-front) the Output Window on completion of the current command. *Show* is a truth. This function may be used in the actions for commands which produce output which the user will want to be made aware of, e.g. **report**.

### *show\_vars*

```
show_vars()
```

Shows all (local and global) user-defined variables and their values. Useful for debugging.

### *show\_window*

```
show_window(window-name, item, workspace, newwindow)
```

Tells ACE to open the specified window (or make it the front window if already open), with *item* as the shown item, if applicable. *Window-name* may be any of the window names returned by [get\\_upfront\\_window\(\)](#). *Item* identifies an item of the required type for the browser windows, as follows:

<b>Part Viewer</b>	part/version
<b>Issue Viewer</b>	version and workspace
<b>CR Viewer</b>	CR number
<b>Baseline Viewer</b>	baseline name
<b>Find</b>	subsystem to look in
<b>Instance Viewer</b>	instance

As the nature of the elements of *item* are unknown, they should *not* be quoted. If *item* is the empty string the browsers will default as if called from the pulldown menus. *Item* is ignored for other windows.

In addition for the Issue Viewer, *workspace* should define the workspace for the issue. *Workspace* is ignored for other windows.

*newwindow* is a true/false indicating whether to open a new viewer window

Note that the name returned may not be the same as that shown in ACE if [nomenclature mappings](#) are in use; for example out-of-the-box maps Issue to Check-out.

**Example**

```
show_window("Part Viewer", var(part), '', true)
```

Opens a new Part Viewer window on the specified part

***sleep***

```
sleep(secs)
```

Sends **AllChange** to sleep for the specified number of seconds.

***socket\_accept***

```
socket_accept(socketnum):
```

Accepts a connection request from a client on the internal socket *socketnum* (which should have been previously created by `socket_server()`).

On successful acceptance, ceases listening on `l` and replaces it by the accepted socket.

***socket\_close***

```
socket_close(socketnum):
```

Closes the internal socket *socketnum*.

***socket\_connect***

```
socket_connect(clientportname, servername, serverportname):
```

Creates a client socket and connects it to a server. *clientportname* specifies the port name/number on the client to use; if this is empty (or 0) a unique port number is used. *servername* specifies the name/IP address of the server; *serverportname* specifies the port name/number on the server to use.

Returns the internal socket number referring to the socket (-1 on error).

***socket\_file***

```
socket_file(socketnum):
```

Attaches a file for i/o to the accepted/connected internal socket *socketnum*. Returns a file handle (a number) (0 on error) which can be used for i/o from/to the socket by file access function calls, cf.

```
fopen().
```

Note, however, that this handle should be closed with `socket_close(socketnum)`, **not** `fclose()`.

***socket\_server***

```
socket_server(portname):
```

Creates a server socket and listens for an incoming connection on it.

*portname* specifies the port name/number to use; if this is 0 a unique port number will be used.

Returns a two element list: the internal socket number referring to the socket (-1 on error), followed by the port number used.

***sortlist***

```
sortlist(lst, func)
```

Returns the new list resulting from sorting list *lst*. If *func* is empty ( ' ') the list is sorted in ascending alphabetical order (case insensitive); " (double quote) characters are ignored, making it suitable for lists of part or file names where the items may be quoted. Any other sort criteria may be used by writing a user-defined function and supplying its name as *func*. The user-defined function will be repeatedly called to compare two items in the list; for speed these are placed in the ACCELvariables `sort1` and `sort2` rather than being passed as parameters to the function. The function should return a number less than, equal to, or greater than 0 (zero) according as `sort1` is to be considered less than, equal to, or greater than `sort2`.

### Example

```
# sort the CRs affecting a part in increasing alphabetical
# order
sortlist(crsaffecting('/part', ''), '')

# sort a list of numbers in increasing numerical order
sortnum          # user-defined function definition
action
  sort1 - sort2;
end

sortlist(var(numbers), 'sortnum')

# sort a file with lines of the form: <word> <number> ...
# primarily in reverse number order,
# secondarily (i.e.\ when numbers are the same)
# in increasing case-insensitive alphabetical order
sortnumword      # user-defined function definition
action
  setvar_list('word1 num1', sort1);
  setvar_list('word2 num2', sort2);
  if var(num1) != var(num2) then
    return var(num2) - var(num1);
  endif;
  if lowercase(var(word1)) > lowercase(var(word2)) then
    return 1;
  endif;
  if lowercase(var(word1)) < lowercase(var(word2)) then
    return '-1';
  endif;
  return 0;
end

sortlist(repcommand('', 'file'), 'sortnumword')
```

### *spell\_engine\_add\_word*

```
spell_engine_add_word(word)
```

Adds the word to the user's dictionary, and saves the updated dictionary to the disk. The user may edit their dictionary in the Misc | Options dialog within the AllChange front-end.

There is no return value.

### Example

```
spell_engine_add_word("NewProject")
```

causes the word NewProject to be added to the user's dictionary.

***spell\_engine\_check\_spelling***

```
spell_engine_check_spelling(text)
```

Returns a list of misspelled words found in the text passed in, or empty if all words are spelled correctly. The list of words is in the order in which the words are found in the text.

**Example**

```
spell_engine_check_spelling("This is some exampe text to check speling")
```

returns a list containing the words "exampe" and "speling".

***spell\_engine\_find\_word***

```
spell_engine_find_word(word)
```

Returns *true* if the word is found in the dictionary, else *false*.

**Example**

```
spell_engine_find_word("exampe")
```

returns *false*.

***spell\_engine\_get\_ignore\_numbers***

```
spell_engine_get_ignore_numbers()
```

Returns whether the user has enabled the option to ignore words with numbers.

***spell\_engine\_get\_ignore\_uppercase***

```
spell_engine_get_ignore_uppercase()
```

Returns whether the spell engine is currently ignoring words consisting only of upper-case characters.

***spell\_engine\_get\_language***

```
spell_engine_get_language()
```

Returns the current dictionary language for the spell engine. See [spell\\_engine\\_set\\_language\(\)](#) for more details.

***spell\_engine\_get\_suggestions***

```
spell_engine_get_suggestions(word)
```

Returns a list of suggestions for an unknown word passed in. If the word passed in is found in the dictionary, then the returned list is empty.

### **Example**

```
spell_engine_get_suggestions("exampe")
```

returns a list containing the words ample, example, exempld, examples, exempt and expel.

### ***spell\_engine\_ignore\_word***

```
spell_engine_ignore_word(word)
```

Adds the word to a list of words to ignore when checking spelling in the current session. Ignored words are not preserved between sessions, and so the word may need to be ignored again in future sessions, or added to the user's dictionary.

There is no return value.

### **Example**

```
spell_engine_ignore_word("NewProject")
```

causes the word NewProject to be ignored by the spell-checker in the current session.

### ***spell\_engine\_is\_ignored\_word***

```
spell_engine_is_ignored_word(word)
```

Returns *true* if the word is a word which is ignored by the spell checker due to the user's options, e.g. "Ignore words with numbers". This does not take into account those words added to the ignore list by `spell_engine_ignore_word`.

### **Example**

```
spell_engine_is_ignored_word("ASCII")
```

Returns *true* if the "Ignore words in UPPERCASE" option is enabled..

### ***spell\_engine\_save\_options***

```
spell_engine_save_options()
```

saves the current spell engine settings to the user's registry profile. This function should only be used in ACE or ACC.

### ***spell\_engine\_set\_ignore\_numbers***

```
spell_engine_set_ignore_numbers(ignore)
```

specifies whether the spell engine should ignore words containing numbers

### ***spell\_engine\_set\_ignore\_uppercase***

```
spell_engine_set_ignore_uppercase(ignore)
```

specifies whether the spell engine should ignore words all in upper-case.

### *spell\_engine\_set\_language*

```
spell_engine_set_language(language)
```

sets the dictionary language to use for the spell engine. The *language* should be one of "English (UK)" or "English (US)". The language defaults to "English (UK)" unless the user's locality is the US.

### *splicelist*

```
splicelist(lst, start, len, replacelst)
```

Replaces the section of list *lst* starting at position *start* of length *len* by *replacelst* ("splices" *lst*). *Start* counts from 0. This can be used to perform a variety of list operations: if *len* == 0 then *replacelst* is inserted into *lst* at *start*; if *start* >= listlen(*len*) then *replacelst* is appended to *lst*; if *replacelst* == '' then *len* elements starting at *start* are deleted from *lst*. Returns the resulting list.

#### Example

```
setvar(lst, '0 1 2 3 4')
setvar(lst, splicelist(var(lst), 1, 2, 'a b'))
# var(lst) == '0 a b 3 4'
setvar(lst, splicelist(var(lst), 1, 2, ''))
# var(lst) == '0 3 4'
setvar(lst, splicelist(var(lst), 1, 0, '1 2'))
# var(lst) == '0 1 2 3 4'
setvar(lst, splicelist(var(lst), 9999, 0, 'end'))
# var(lst) == '0 1 2 3 4 end'
```

### *split*

```
split(str, splitter)
```

Returns the list resulting from splitting string *str* on the first character of *splitter*. If *splitter* is the empty string split on whitespace (space or tab).

#### Example

```
split('1-2-3', '-')
returns 1 2 3
```

### *splitquote*

```
splitquote(str, splitter)
```

Like `split()`, but respects token quoting within *str* so that a token is never split. Use `splitquote()` if *str* holds filenames which may contain spaces and thus be quoted. When ACCEL automatically converts a string to a list this is equivalent to `splitquote(str, ' ')`.

#### Example

```
split('a "b c" d', ' ') → 'a', '"b', 'c"', 'd' (4 items)
splitquote('a "b c" d', ' ') → 'a', '"b c"', 'd' (3 items)
```

### ***splitunquote***

`splitunquote(str, splitter)`

Like `splitquote()`, but removes quotes from elements in the resulting list.

#### **Example**

```
splitquote('a "b c" d', ' ') → 'a', '"b c"', 'd'
splitunquote('a "b c" d', ' ') → 'a', 'b c', 'd'
```

### ***status\_is\_open***

`status_is_open(class, status)`

Returns whether the status in the cycle associated with *class* is "open", e.g. to display only "open" CRs in the CR browser use:

```
status_is_open(cr_class, cr_status)
```

in the condition box.

### ***status\_list***

`status_list(cycle)`

Returns a list of the statuses in cycle.

### ***strlen***

`strlen(str)`

Returns the length of string *str*.

#### **Example**

```
strlen('0123')
```

Returns 4

### ***subst***

`subst(string, search, replace)`

Returns the string resulting from substituting all occurrences in *string* of *search* by *replace*.

#### **Example**

```
subst('Do cats eat catfish?', 'cat', 'dog')
```

Returns Do dogs eat dogfish?

```
subst('this has no spaces', ' ', '')
```

Returns thishasnospaces

### ***sys\_date***

`sys_date(format)`

Returns a string containing the date/ time (or some portion of it) according to the format given by the string *format*. The format may be any of:

<b>all</b>	the full date and time as <code>yyyy/mm/dd hh:mm:ss</code>
<b>all2</b>	the full date and time as <code>dd/mm/yyyy hh:mm:ss</code>
<b>allus</b>	the full date and time as <code>mm/dd/yyyy hh:mm:ss</code>
<b>date</b>	the date as <code>yyyy/mm/dd</code>
<b>date2</b>	the date as <code>dd/mm/yyyy</code>
<b>dateus</b>	the date as <code>mm/dd/yyyy</code>
<b>time</b>	the time as <code>hh:mm:ss</code>
<b>windate</b>	windowing system specified date format (short).
<b>winlongdate</b>	windowing system specified date format (long).
<b>wintime</b>	windowing system specified time format.
<b>winall</b>	windate followed by wintime
<b>year</b>	the year as <code>yyyy</code>
<b>year4</b>	the year as <code>yyyy</code>
<b>year2</b>	the year as <code>yy</code>
<b>month</b>	the month as <code>mm</code>
<b>day</b>	the day as <code>dd</code>
<b>hour</b>	the hour as <code>hh</code>
<b>minute</b>	the minute as <code>mm</code>
<b>second</b>	the seconds as <code>ss</code>

windate, winlongdate, wintime and winall take into account a windowing system preference for date formats. Under Windows this is the standard date format (as specified via Control Panel).

In order to create your own date/ time formats, repeated calls of the `sys_date` function will provide each portion of the date/ time as required.

### Example

```
echo('The current year is '.sys_date(year))
```

### takeout

```
takeout(args, part, workfile)
```

Calls the VC **Takeout** tool with arguments *args* on the VC file corresponding to *part*. *Workfile* specifies the path for the extracted workfile: if it is `' '` the default will be used (i.e. same name as VC file and situated in current workspace directory hierarchy, and it will left read-only); if it is `none` no workfile will be created and is used to indicate that the no get command line option (`-g`) is used.

For information on valid arguments see the **AllChange VC Tools** manual.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

### takeout\_for\_edit

```
takeout_for_edit(args, part, workfile)
```

Calls the VC **Takeout** tool with arguments *args* on the VC file corresponding to *part* to do a takeout for edit. *Workfile* specifies the path for the extracted workfile: if it is `' '` the default will be used (i.e. same name as VC file and situated in current workspace directory hierarchy); if it is `none` no workfile will be created.

For information on valid arguments see the **AllChange VC Tools** manual.

This function *must* be used instead of `command`.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

***temp\_dirname***

```
temp_dirname(dirname, prefix)
```

Returns a unique directory name intended for temporary use. *Prefix* gives the basename of the directory; 3 digits will be appended to this. The directory specified by *dirname* — if not empty — is prepended to the new directory name. The directory is actually created; the implementation guarantees that it will not have existed beforehand. Every call to `temp_dirname()` generates and creates a new temporary directory, so the result should be saved in a variable. The code should ensure any temporary directories created are deleted.

**Example**

```
traperror
    setvar(mytmpdir, temp_dirname(tmpdir, 'TMP')); # creates a temporary
directory in tmpdir named TMP
    setvar(tmpfil, temp_filename(var(mytmpdir), 'TMPFIL', 'txt')); #
create a temporary file in the temporary directory
    writefile(var(tmpfil), false, true, "Test Line to Temp File"); #
replace this with whatever processing is needed for the temporary file
onerror
    # On Error occurring if temporary directory created delete it and all
its contents
    if var(mytmpdir) != '' then
        deldir(var(mytmpdir), true);
    endif;
    error(error_message);
enderror;
# If temporary directory created delete it and all its contents as fin-
ished with it now
if var(mytmpdir) != '' then
    deldir(var(mytmpdir), true);
endif;
```

***temp\_filename***

```
temp_filename(dirname, prefix, suffix)
```

Returns a unique filename intended for temporary use. *Prefix* gives the basename of the file; 3 digits will be appended to this. *Suffix* gives the suffix of the file: if it is not empty it will be appended (preceded by a `.`) to the filename. The directory specified by *dirname* — if not empty — is prepended to the filename. The file is actually created; the implementation guarantees that it will not have existed beforehand. Every call to `temp_filename()` generates and creates a new temporary file, so the result should be saved in a variable. The code should ensure any temporary files created are deleted.

**Example**

```
setvar(file, temp_filename(tmpdir, 'Word', 'doc'))
```

Might set the variable `file` to `c:\Temp\Word001.doc`, and creates this file

***top\_instance***

```
top_instance(version)
```

Return the top instance of the specified *version*, where the top instance is defined as the instance with the highest number.

**Example**

```
top_instance('/Car/CD Player;004')
```

Might return /Car/CD Player;004:23 if this is the highest numbered instance of version 004 of /Car/CD Player

***trace***

```
trace(name)
```

Adds *name* to the list of ACCEL user-defined variables/ functions to be traced — see [Debugging](#). If *name* is `all` variables and functions are traced; if *name* is `functions` then all functions are traced; if *name* is `none` all variable and function tracing is switched off. Tracing must be enabled via `set trace=on`. Useful for debugging.

**Example**

```
trace('MyFunction')
```

Starts tracing calls to MyFunction

```
trace('functions')
```

Starts tracing all calls to all functions

***trace\_echo***

```
trace_echo(string)
```

Outputs *string* to the trace output — see [Debugging](#). Tracing must be enabled via `set trace=on`. Useful for debugging.

***trace\_echo\_map***

```
trace_echo_map(string)
```

As [trace\\_echo](#) function, except that any nomenclature [mapping markers](#) in *string* are translated to the current nomenclature.

**Example**

```
trace_echo_map("#Part# name is ".pa_part)
```

Will echo into the output/debug windows when trace is switched on, when the term Part is mapped to the term CI (Configuration Item) 'CI name is <name of ci>'

***trim\_spaces***

```
trim_spaces(str)
```

Returns *str* with all leading and trailing whitespace removed.

***unc\_path***

```
unc_path(path)
```

Tries to convert *path* to a UNC (`\\machine\share\...`). Returns the resulting path. If this fails for any reason (e.g. *path* does not start with *drive:*) returns *path* unchanged.

### *unformat\_number*

```
unformat_number(<number>)
```

Converts a local format number to Intasoft internal format <num>.<num>

#### **Example**

```
unformat_number("1,23") will produce "1.23"
```

### *unformat\_date*

```
unformat_date(string)
```

Accepts a string in a variety of date formats and returns a date string in the format produced by `format_date(date-string, 'date')`, i.e. `yyyy/mm/dd`. This may then be used in contexts which require this date format. The input string should contain a day, month and year, separated by whitespace or punctuation; these may come in (almost) any order, and the month may appear as a word.

If the windowing system preference for date formats is U.S. style, *string* is interpreted as `mm/dd/yyyy` in preference to `dd/mm/yyyy`.

If the year part is specified as a 2 digit number then this is treated as 19yy if yy is greater than or equal to 70 and as 20yy otherwise.

#### **Example**

```
unformat_date('January 1, 1997') == '1997/01/01'  
unformat_date('2001 Dec 31') == '2001/12/31'  
unformat_date('97/2/1') == '1997/02/01'  
unformat_date('1/2/97') == '1997/02/01'  
unformat_date('2001/02/03') == '2001/02/03'  
unformat_date('01-02-03') == '2003/02/01'
```

### *unprotfile*

```
unprotfile(filename)
```

Unprotects the file *filename* (i.e. makes the file writable).

### *unprotserverfile*

```
unprotserverfile(filename)
```

If running client/server gets the server to unprotect (make writable) a file; *filename* should specify a path from the server's point of view. If not running client/server acts the same as `unprotfile()`.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

### *unquote*

```
unquote(str)
```

Undoes the effect of `quote()`, in case this is ever needed.

### *unquote\_arg*

```
unquote_arg(str)
```

Undoes the effect of `quote_arg()`, in case this is ever needed.

### ***unquote\_arglist***

`unquote_arglist(list)`

Undoes the effect of `quote_arglist()`, in case this is ever needed.

### ***unquote\_os***

`unquote_os(str)`

Undoes the effect of `quote_os()`, in case this is ever needed.

### ***unquote\_oslist***

`unquote_oslist(list)`

Undoes the effect of `quote_oslist()`, in case this is ever needed.

### ***unquotelist***

`unquotelist(list)`

Undoes the effect of `quotelist()`, in case this is ever needed.

### ***update\_bt\_file***

`update_bt_file(bt-filename)`

Updates the Build Thread file *bt-filename* to remove any "EDIT" marks for parts that are no longer checked out for edit to the current workspace/ user. It may only be used in the actions of the **promote** command and checks that the part(s) have been returned from edit by the **promote**.

### ***update\_progressbar***

`update_progressbar(howmany, total)`

Updates the progress bar to show the percentage: *howmany* divided by *total*. Creates a fresh progress bar if no progress bar exists. The functions `num_records()` and `num_records_in_range()` may be useful for measuring progress through a database (though if using an `each` loop this can be accomplished automatically by using the `-progress` flag).

This function is supported for both the Windows interface and the browser interface.

### ***update\_statusbar***

`update_statusbar(message)`

Updates the statusbar to show *message*.

This function is supported for both the Windows interface and the browser interface.

### ***uppercase***

`uppercase(str)`

Returns string *str* converted to upper case.

### *usereql*

```
usereql(user1, user2)
```

Returns whether two usernames are "equal" (i.e. the same). Under Windows this is case insensitive.

### *user\_to\_fullname*

```
user_to_fullname(username)
```

Converts a user login name/ID to the user's full/friendly name as defined in the user registrations. If no full name has been specified for the user, the name passed is returned.

#### **Example**

If the user registrations define that user name 'fred' has a full name of 'Fred Bloggs' then:

```
user_to_fullname('fred') returns 'Fred Bloggs'
```

### *user\_to\_fullnamelist*

```
user_to_fullnamelist(username)
```

Converts a list of user login name/IDs to full/friendly names as defined in the user registrations. If no full name has been specified for a user, the name passed is returned.

#### **Example**

If the user registrations define that user name 'fred' has a full name of 'Fred Bloggs' and user named jon has the full name of 'Jonathan Smith' then:

```
user_to_fullnamelist('fred jon')
```

Returns 'Fred Bloggs' 'Jonathan Smith'

### *utc\_to\_localtime*

```
utc_to_localtime(date-time)
```

Converts *date-time* from UTC (GMT) time to local time.

*Date-time* should be specified in standard Intasoft format (YYYY/MM/DD hh:mm:ss).

The string returned is in standard Intasoft format.

#### **Example**

```
utc_to_localtime("2002/10/11 00:00:00")
```

Will return "2002/10/10 19:00:00" if you are in EST

### *valid\_voters*

```
valid_voters(database, item, status)
```

Returns a list of the users who could potentially cast a vote for *item* in *database* in *status*.

#### **Example**

```
valid_voters('cr', 'RFC00010', 'CAB')
```

Might return Sarah, James and Joe if these are the valid voters for RFC00010 in CAB status

### ***var***

```
var(var-name)
```

Returns the value of the user-defined variable named *var-name*.

### ***vccommand***

```
vccommand(cmd, cmdline)
```

Issues an arbitrary VC command with an arbitrary command-line. Returns the command's exit code. This does not run the command client/server, even if running VC client/server. This function is for use by administrator only. It should be used instead of `command()` for correct behaviour.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

### ***vcdir\_exists***

```
vcdir_exists(dirname)
```

Returns whether VC directory *dirname* exists.

This function *must* be used instead of `dir_exists()` if running VC client/server.

### ***vcerase***

```
vcerase(args, part)
```

Calls the VC **Vcerase** tool with arguments *args* on the VC file corresponding to *part*.

This function *must* be used instead of `command`.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

### ***vcfile\_exists***

```
vcfile_exists(filename)
```

Returns whether VC file *filename* exists.

This function *must* be used instead of `file_exists()` if running VC client/server.

### ***vcinfo***

```
vcinfo(args, vcfile, outputfile)
```

Calls the VC **Vcinfo** tool with arguments *args* on the VC file *vcfile*. *Outputfile* specifies the path to a file into which the output from the **Vcinfo** will be placed; if it is ' ' the output will be sent to the Command Output window.

### ***vcrep***

```
vcrep(args, vcfile, outputfile)
```

Calls the VC **Vcrep** tool with arguments *args* on the VC file *vcfile*. *Outputfile* specifies the path to a file into which the output from the **Vcrep** will be placed; if it is ' ' the output will be sent to the Command Output window.

### ***vcscan***

```
vcscan(args, vcfile)
```

Calls the VC **Vcscan** tool with arguments *args* on the VC file *vcfile*. Returns true if *vcfile* contains the string passed at the end of *args*, false if it does not.

### *version\_list*

`version_list(part, what-versions)`

Returns a list of the versions of the part. The part should be given as a full path name. *What-versions* determines which versions are to be included in the list. This may be any of:

- all** all versions will be included, excluding those reserved but not actually representing a version (have not been created as a result of and check out for edit)
- def** only the default version will be included
- edit** all versions will be included including those reserved
- reg** the registered version matching the part is returned, or the top-most if there is no matching registration. This excludes any check-outs in the current workspace.
- top** the top-most version matching the part is returned. This excludes any check-out or registrations in the current workspace.

### *version\_successor*

`version_successor(version)`

Returns the direct successor version to *version*, or '' if none exists. "Direct" means the successor version must lie on the same trunk/ branch as *version*. This will never return new versions which were created as result of optimistic locking.

### *version\_template\_text*

`version_template_text(part-class)`

Returns, as a list of lines, the text of the template file, if any, for new versions of class *part-class*.

### **Example**

```
version_template_text('Source')
```

Returns the contents of the template file for versions of part of class 'Source', this might be for example:

```
Version Notes::  
Implementation Action::
```

### *voters\_outstanding*

`voters_outstanding(database, item, status)`

Returns a list of the possible users still to cast a vote for the current vote in *database* for *item* in *status*.

### **Example**

```
voters_outstanding('cr', 'RFC00010', 'CAB')
```

Might return Joe and James as voters still to vote on RFC00010 in status CAB if they are both voters for RFC00010 and have not yet voted .

### *votes\_cast*

`votes_cast(database, item, status, votestatus, votenumber):`

Returns the number of votes cast in *database* for the *item* in *status* for which the voted status was *votestatus*. The *votenumber* parameter specifies which zero-indexed set of voting to visit, where 0 is the first vote set. The current or latest vote set may be specified by using the value '-1', and all vote sets may be specified by using the value '-2'.

See also [num\\_votes](#).

### Example

```
votes_cast('cr', 'RFC00010', 'CAB', 'Approved', '-1')
```

Might return 2 if 2 votes have been cast for Approved on RFC00010 in status CAB in the current vote set

### *votes\_outstanding*

`votes_outstanding(database, item, status)`

Returns the number of votes still possible for the current vote in *database* for *item* in *status*. If there is no current open vote, the empty string is returned.

### Example

```
votes_outstanding('cr', 'RFC00010', 'CAB')
```

Might return 1 if one voter remains to cast a vote on RFC00010 in status CAB.

### *vote\_deadline*

`vote_deadline(database, item, status )`

Returns the actual deadline date/time for *item* in *status*. The deadline is returned in local time, in the Standard Intasoft (YYYY/MM/DD hh:mm:ss) format.

### Example

```
vote_deadline('cr', 'RFC00010', 'CAB')
```

Might return 2008/07/05 12:00:00 if the deadline for the vote is defined as 1 day after the vote start date and the vote start date was 2008/07/04 12:00:00.

### *vote\_decision*

`vote_decision(database, item, status, votenumber)`

Returns the decision reached in the vote set specified by *votenumber* in *database* for *item* in *status*.

The *votenumber* parameter specifies which zero-indexed set of voting to visit, where 0 is the first vote set. The current or latest vote set may be specified by using an index of '-1', and all vote sets may be specified with an index of '-2'.

The returned string will be the decision status if a conclusive decision was able to be reached, "BLOCKED" if the vote was blocked, 'any' if the vote is an advisory vote, or an empty string if the specified vote does not exist or is still open.

See also [num\\_votes](#).

## Example

```
vote_decision('cr', 'RFC00010', 'CAB', '-1')
```

Might return 'Approved' if the latest vote on RFC00010 in CAB status resulted in the Approved status being the result of the vote.

## *vote\_initial\_mail\_users*

```
vote_initial_mail_users(database, item, status)
```

Returns a list of the users to whom the initial vote mail will be sent for *item* in *database* in *status*. For a serial vote, this function will return only the first voter who has not yet cast a vote.

## Example

```
vote_initial_mail_users('cr', 'RFC00010', 'CAB')
```

If there is a vote defined on the CAB status with voters as the content of the CAB field and Jim, Joe and Sarah are the users selected on the CAB field of RFC00010.

If the vote is a non serial vote the function would return Joe, Jim and Sarah

If the vote is defined as a serial vote and Joe has passed the vote to Jim, then the function would return Jim

## *vote\_start\_date*

```
vote_start_date(database, item, status, votenumber)
```

Returns the start date/time of the vote set specified by *votenumber* for the *item* in *status*. The *votenumber* parameter specifies which zero-indexed set of voting to visit, where 0 is the first vote set. The current or latest vote set may be specified by using an index of '-1'. The start date is returned in local time, in the Standard Intasoft (YYYY/MM/DD hh:mm:ss) format.

## Example

```
vote_start_date('cr', 'RFC00010', 'CAB', '-1')
```

Might return 2008/07/05 12:00:00 if the vote on RFC00010 in CAB status started on that date.

## *wild\_dirs*

```
wild_dirs(dirpattern)
```

Returns a list of directories matching the operating system directory pattern, e.g. `c:\public\*.*`.

## *wild\_files*

```
wild_files(filepattern)
```

Returns a list of files matching the operating system file pattern, e.g. `c:\public\*.out`.

## *wild\_parts*

```
wild_parts(partpattern)
```

Returns a list of parts matching the part pattern, e.g. `/subsys/*.c`. *Partpattern* may refer to versions, e.g. `*.c`; (default version) or `baseline!*;*` (all versions in baseline).

### *wild\_server\_dirs*

`wild_server_dirs(dir, dirpattern)`

If running client/server gets the server to return a list of directories in *dir*; *dir* should specify a path from the server's point of view; if *dirpattern* is not empty only directories matching it are returned. If not running client/server acts the same as `wild_dirs()`.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

### *wild\_server\_files*

`wild_server_files(dir, filepattern)`

If running client/server gets the server to return a list of files in *dir*; *dir* should specify a path from the server's point of view; if *filepattern* is not empty only files matching it are returned. If not running client/server acts the same as `wild_files()`.

This function is classed as "dangerous" — see ["Dangerous" Functions](#).

### *wild\_vars*

`wild_vars(varnamepattern)`

Returns a list of existing user-defined variables whose name matches the specified pattern, e.g. `Option_*`.

### *work\_filename\_part*

`work_filename_part(path)`

Returns the part corresponding to *workfilepath* in the current workspace. The user must be attached to a workspace for this to work.

### *writefile*

`writefile(filename, append, newline, string)`

Writes *string* to operating system file *filename*. If *append* is set to true the string is appended to the file; if it is set to ' ' (false) it rewrites the file from the beginning. If *newline* is set to true a newline is added at the end of the string; if it is set to ' ' (false) no newline is added.

### *yn*

`yn(str)`

If the string given as the parameter is true (not empty), this function returns the string `yes`, otherwise it returns the string `no`. A truth value as the parameter supplies a suitable argument.

## User-defined Functions

In addition to all the predefined functions ACCEL permits user-defined functions to be written. User-defined functions can range in scope from one-line utility functions to aid command definition writing to large bodies of code offering complex high-level operations available to the user from a pull-down menu in ACE.

User-defined functions may be written in the command definitions file (see [Command Definitions](#)) or in any file read in via the ACCEL `read_functions` function. Whichever file they are read from they have the form:

```
func-name
    action
    ACCEL statements
end
```

and are separated from one another by a blank line.

User-defined functions must always be called using the `call` function; this accepts a variable number of parameters, the first of which is the function name and the remainder any parameters to pass to the function, i.e.:

```
call(func-name, param1, param2, ...)
```

Any parameters are accessible within the function as user-defined (local) variables (see [User-defined Variables](#)) `p1, p2, ...`, e.g.:

```
echo(var(p1) . ' ' . var(p2));
```

User-defined functions must always be called with the correct number of parameters. The parameters to a user-defined function are local variables in that function; further variables may be declared local to the function via the `local` function. Global variables may be read and set as normal.

User-defined functions may call other user-defined functions, including themselves recursively. They may return a result to the caller: the result of a user-defined function is the last statement executed. They may return explicitly or prematurely, with or without a result, by using the `return` statement.

User-defined functions designed to be called directly by the user may be added into the Front-end menus as described in [Menu Items](#). Selecting such a function from ACE will cause a command of `eval call(function-name)` to be issued. Any information required from the user may be prompted for using the ACCEL `prompt... functions`.

Local variables may be defined for user defined functions using the `local` built in function, e.g.:

```
local(var1, var2)
```

Note that the parameters received by a function do not need to be declared as local. However it is good practice and helps in the readability of ACCEL code if a local variable is declared for each parameter with a meaningful name for the parameter, and then this assigned to the appropriate `p1, p2` variable. All further references to the parameters in the function should then be via the meaningful variable name.

### Example

```
MyFunc
action
  local(Firstname, Surname);
  setvar(Firstname, var(p1));
  setvar(Surname, var(p1));

  echo("Hello ".var(Firstname)." ".var(Surname));
end
```

### "Dangerous" Functions

Certain predefined ACCEL functions are classed as "dangerous", meaning that an end-user could potentially use them to perform an operation which he should not usually be allowed to do. In addition, some command-line arguments to certain **AllChange** commands are classed in the same way. In both cases, however, the Administrator may need to write ACCEL code which will execute on behalf on the user to perform such an operation.

Both of these potentially "dangerous" operations may only be performed provided the ACCEL code calling the function or executing the command originates from the **AllChange** project, template or system directory, or the current user has set his role to **dbsuperuser**. This is known as a *secure context*.

In essence this means that code which wishes to call a "dangerous" ACCEL function (e.g. `fopen_in_projdir()`) or execute a "dangerous" command (e.g. `alterversion symname=...`) must be situated in a command action or user-defined function which has been read in from `commands.ac`, `cycles.ac`, `cmdfunc.ac` or other such file which lives in the project, template or system directory. End-users will get an error if they

try to perform one of these operations directly from ACE , or from any ACCEL function they might write for themselves. However, to aid development, debugging etc. administrative users *may* perform such an operation, provided they set their role to **dbsuperuser**.

There is also a predefined ACCEL variable, `secure_context`, which returns whether the current ACCEL code is executing in a secure context, as defined above. Since this variable is not set when executing command/ life-cycle conditions, but is set during command/ life-cycle actions, this can be used in an entry condition to test whether the command has been called from actions in another command, as opposed to directly by the user. This addresses a common requirement: a (normal) field should be alterable in ACCEL code read from, say, `commands.ac` in response to something the user is doing, but it should not be *directly* alterable by the user.

Example (extract from `commands.ac`):

```
alterbaseline
entrycond
  if getarg('arb1') then
    if not secure_context then
      error('End-users may not alter this field');
    endif;
  endif;
end

baseline
action
  db_tran_flush(); # flush before changing self
  interpret('alterbaseline arb1="This is a non-user-alterable field" '.bl_name);
end
```

The list of ACCEL functions which are "dangerous" is currently: `add_status_log`, `alter_status_log`, `control`, `copyserverfile`, `copyvcfile`, `createserverdir`, `createvmdir`, `delservdir`, `delservfile`, `del_status_log`, `delvmdir`, `delvcfile`, `fopen_in_projdir`, `fopen_in_sysdir`, `getserverfile`, `protserverfile`, `putaway`, `putserverfile`, `renamecr`, `renameversion`, `renserverfile`, `renvcfile`, `takeout`, `takeout_for_edit`, `unprotserverfile`, `vccommand`, `vcerase`, `wild_server_dirs`, `wild_server_files`

Some **AllChange** commands accept "dangerous" command-line options. These allow ACCEL code to be written to alter certain normally read-only fields — or fields which are normally altered by other commands — of databases, as follows:

- **add & alterstatus** =...
- **baseline & alterbaseline** design=... date=... status=... user=...
- **newcr & altercr** assignee=... status=... originator=... number=...
- **newversion & alterversion** date=... status=... symbname=... user=...
- **alterversion** pred\_ver=...

Note that when setting a field which would normally be altered by another command the standard processing is not performed. So, if the assignee is changed neither the condition nor actions of the **assigncr** command are performed; if the status is changed no conditions/ actions of the **status** command *or* of the old or new status in the cycles file are performed, nor is the usual check on the validity of the progression executed. This is very useful for getting the databases into a desired state, but should be used with care.

## ACCEL Variables

### User-defined Variables

In addition to all the predefined variables ACCEL permits user-defined variables to be created. These may be used for any purpose.

User-defined variables may be assigned a value by the following built in functions:

- [setvar](#)
- [setvar list](#)
- [appendlistvar](#)
- [appendstrvar](#)

Their value is read by the [var](#) function, e.g.

```
setvar(count, 0);  
setvar(count, var(count) + 1);
```

Any value assigned to them retains its data type; it may be converted to any type when it is read, according to context.

User defined variables are created as referenced and initialised to the empty string.

User-defined variables are always global in scope; this means that a variable defined in one function may be accessed from another called by the first function even if the variable was declared local.

variables are global in life unless declared local using the [local](#) built in function. This means that variables defined as local may not be accessed once the function exits. As a matter of good practice we recommend naming all global variables as `Global_varname` and all others should be declared local.

Parameters to [user-defined functions](#) are received as local variables.

## Built In Variables

### *About ACCEL Built In Variables*

A number of global built in variables are available within ACCEL. Some variables only contain a valid value under certain circumstances, as indicated below; if used incorrectly results may be unpredictable, so care should be taken of their usage.

#### **A-Z list of ACCEL Variables**

##### **acc**

This variable is true when the ***AllChange*** command line system is being used and false when ACE is being used.

##### **acexedir**

The full path of the ***AllChange*** executables directory.

##### **achelpdir**

The full path of the ***AllChange*** on-line help-files directory.

##### **achomedir**

The full path of the user's ***AllChange*** home directory. This will be the environment/.ini variable `ACHOME` if there is one, otherwise their real home directory (`homedir`).

##### **acprojdir**

The full path of the ***AllChange*** project directory.

##### **acproject**

The name of the current ***AllChange*** project.

**acprojtemplate**

The name of the project template used in the creation of the current **AllChange** project.

**acprojtemplatedesc**

The description of the current **AllChange** project template

**acprojtemplatedir**

The full path to the current **AllChange** project template directory

**acregversion**

The current **AllChange** version number used to determine where in the registry its settings are stored. The root path used is: `HKEY_CURRENT_USER\Software\Intasoft\AllChange\<acregversion>`

**acserverprojdir**

If running client/server, returns the full path of the **AllChange** project directory from the server's point of view (i.e. the value of the `ACSERVERPROJDIR` environment variable passed from the client to the server). If not running client/ server, returns the full path of the **AllChange** project directory (i.e. same as `acprojdir`).

**acserversysdir**

If running client/server, returns the full path of the **AllChange** system directory from the server's point of view. If not running client/server, returns the full path of the AllChange system directory (i.e. same as `acsysdir`).

**acsqldb**

The name of the SQL server database used for this project.

**acsqserver**

The name of the SQL Server/instance being used.

**acsysdir**

The full path of the **AllChange** system directory.

**acversion**

The current **AllChange** version number. Useful when writing code which uses certain features.

**acweb**

Returns true if running as part of the web interface.

**admin\_users**

A list of the **AllChange** administrative users. If the [Show only project users in lists](#) configuration option is selected then only those users which are valid for the project and administrative users will be in the list, otherwise all registered administrative users will be in the list.

**allow\_optimistic\_locking**

Returns the value of the **Allow optimistic locking** configuration option

### **any**

Returns any. May be used as a parameter to those functions which place a special meaning on "any", e.g. `is_issued()`.

### **client\_server**

Whether **AllChange** is running client/ server.

### **cmd\_result**

The result (i.e. exit code) returned by the operating system for the last invocation of the `command` function.

### **command\_first\_time**

This variable is only valid during command entry conditions. It returns whether this is the first time a condition is being checked in response to either a top-level user action or the `ACCEL interpret` function. It can be tested in order to do something once prior to executing a command on a number of items or multiple commands on one item (such as prompting for user input). In the case of the `status`, `statusbaseline` and `statuscr` commands `command_first_time` is true only during the command's entry condition (in `commands.ac`), not during the entry/ exit condition of the actual status progression (in `cycles.ac`).

### **command\_items**

A list of the items passed to the current command. Valid only for commands which accept multiple items (e.g. `add` but not `newcr`). Depending on the command the items may or may not have undergone wildcard expansion and/ or parts/ paths been made absolute.

### **command\_line**

The command line used to invoke the current command excluding the command name.

### **command\_name**

The name of the command currently being interpreted.

### **comment**

The comment normally associated with a **return from edit**; it is also set when **newversion** or **alter-version** are used to create a new version. This variable is only valid when one of these commands is invoked.

### **cwd**

The full path for the current working directory.

### **cwp**

The full path giving the current working part.

### **date\_format\_us**

Returns whether the windowing system preference for date formats is U.S. style (mm/dd rather than dd/mm). Under Windows this is taken from the standard date format (as specified via Control Panel).

### **dde\_result\_str**

The string returned from the last DDE request to a DDE server using the `call_dde` function.

**dll\_result\_str**

The string returned, if any, from the last `call_dll()`. The supplied `ACINTDLL.C` file shows how a DLL may pass back a string to **AllChange**.

**error\_message**

This variable contains the text of the last error message. It is valid only after an error has been caught in a `traperror` statement, or when ACE has executed a command as a DDE server.

**false**

Returns *false*. Equivalent to '' (empty string) in any truth context, but may be clearer when passing parameters, etc. Note: as with other in-built ACCEL variables, this must *not* be quoted: `false != 'false'`, since 'false' (or "false") is a non-empty string which evaluates to *true*.

**frompathname**

The full path name of the object associated with an existing part. This variable is only valid when the **alter** (when changing the location), **copy**, **delete** or **rename** commands are being invoked.

**generate\_depends**

True if calling **build** to generate the `depends.ac` file, false if calling **build** to actually do the build. Valid only during the actions for the **build** command.

**group\_list**

Returns a list of all group names that are defined.

**homedir**

The full path of the user's real home directory used by **AllChange**.

**interpret\_success**

True/ false according as the last **AllChange** command interpreted (whether directly or through the `interpret()` function) did or did not "succeed", i.e. did not result in an error.

**is\_admin\_user**

Returns whether the current user is an Administrative user.

**is\_cr\_only\_user**

Returns whether the current user is licensed as a CR-only user.

**is\_licensed\_user**

Returns whether the current user is a licensed AllChange user (as opposed to an unlicensed, read-only user — see "Unrestricted Read-only Access to AllChange").

**is\_readonly**

Returns true if the user is unlicensed, or if **AllChange** is run with the `-readonly` command-line argument. Otherwise returns false.

### **item**

The current item in a list when reporting on, or iterating through, each item in a list instead of each item in a database. Saved and restored across nested control levels or `each` loops (just like the current record in a database). Valid only when the **report** command is invoked or in an `each` loop.

### **licensed\_cr\_only\_users**

A list of the permissible CR-only **AllChange** users. If the [Show only project users in lists](#) configuration option is selected then only those users which are valid for the project and CR-only will be in the list, otherwise all registered CR-only users will be in the list.

### **licensed\_users**

A list of the permissible **AllChange** users. If the [Show only project users in lists](#) configuration option is selected then only those users which are valid for the project will be in the list, otherwise all registered users will be in the list.

### **max\_licensed\_cr\_only\_users**

Returns the maximum number of CR-only users allowed by the **AllChange** licence.

### **max\_licensed\_users**

Returns the maximum number of users allowed by the **AllChange** licence.

### **monitoritem**

The name of the item that is to be monitored. This variable is only valid when a monitor is being triggered.

### **need\_promote**

True if the current object should be promoted, false otherwise. Only valid when the **promote** command is invoked.

### **need\_return**

True if the current object should be returned, false otherwise. Only valid when the **promote** command is invoked.

### **newstatus**

The new status requested in a status change command. This variable is only valid when the **status**, **statusbaseline** or **statuscr** commands are invoked.

### **no\_baseline\_baselines\_affected**

This variable returns whether baselines may have baselines affected associated with them (as set by the **Enable baseline affected** configuration option). It is used in command definitions.

### **no\_baseline\_files\_affected**

This variable returns whether baselines may have files affected (attachments) associated with them (as set by the **Enable baseline file attachments** configuration option). It is used in command definitions.

**no\_baseline\_votes**

This variable returns whether baseline voting is disabled (as set by **Allow baseline votes** [configuration option](#))

**no\_bts**

This variable returns whether build threads are disabled (as set by the **Use build threads** configuration option). It is used in command definitions.

**no\_cr\_baselines\_affected**

This variable returns whether CRs may have baselines affected/ solved associated with them (as set by the **Enable baselines affected/ solved** configuration option). It is used in command definitions.

**no\_cr\_crs\_affected**

This variable returns whether CRs may have other CRs affected associated with them (as set by the **Enable CRs affected/ solved** configuration option). It is used in command definitions.

**no\_cr\_files\_affected**

This variable returns whether CRs may have files affected (attachments) associated with them (as set by the **Enable file attachments** configuration option). It is used in command definitions.

**no\_cr\_parts\_affected**

This variable returns whether CRs may have parts affected/ versions solved associated with them (as set by the **Enable partsaffected/ solved** configuration option). It is used in command definitions.

**no\_cr\_votes**

This variable returns whether cr voting is disabled (as set by **Allow cr votes** [configuration option](#)).

**no\_instances**

This variable returns whether instances are disabled (as set by **Allow Instances** [configuration option](#)).

**no\_monitors**

This variable returns whether monitors are disabled (as set by the **Enable monitors** configuration option). It is used in command definitions.

**no\_part\_parts\_affected**

This variable returns whether parts may have other parts affected associated with them (as set by the **Enable parts affecting parts** configuration option). It is used in command definitions.

**no\_part\_votes**

This variable returns whether part voting is disabled (as set by **Allow part votes** [configuration option](#))

**not\_applicable**

This variable returns the literal string "(Not Applicable)". It is used in prompt lists, and allows accurate comparison of selected values.

### **object**

The name of the object currently being built, promoted or released: this is the full pathname for the file. This variable is only valid when the **build**, **promote** or **release** commands are invoked.

### **oldstatus**

This variable is similar to the `newstatus` variable. It contains the name of the status being moved *from* in a status change command. This variable is only valid when the **status**, **statusbaseline** or **statuscr** commands are invoked.

### **ole\_result\_str**

The string returned, if any, from the last `call_ole()`.

### **opsys**

Name of the primary operating system on which **AllChange** is running. Value is `msopsys` for Windows 95/98 and Windows NT.

### **opsys2**

Names the operating system variant. Values are `solaris` for Sun Solaris 2.x, `hpux` for HP-UX and `win32` for 32-bit Windows 95/98 and NT.

### **pool\_object**

The name of the promoted object currently being promoted or released: this is the full pathname to the object in the pool to which it is being promoted or from which it is being released. This variable is only valid when the **promote** or **release** commands are invoked.

### **prev\_index**

The value of the indexed field of last element reported on. This is useful when using break levels in reports. This variable is only valid when the **report** command is invoked.

### **process\_id**

The current process's process id. This may be useful in constructing unique temporary file names

### **putaway\_cmd\_result**

The result (i.e. exit code) returned by the operating system for the last invocation of the `putaway` function. This is useful for determining whether no new version was created because the "do not putaway when no differences" flag on the VC file has been set and no differences were encountered (if special processing is desired in this case).

### **read\_perms\_denied\_message**

The reason for denial of the last unsuccessful call to `is_field_readable()`. If the last call to the `is_field_readable` function was successful then this variable contains the empty string.

### **rep\_command\_line**

The command line used to generate a report. This is the result of the actual command line invocation and the command line arguments set in the format file. This variable is only valid when the **report** command is invoked.

**return\_from\_edit**

True if the current object should be returned from edit, false otherwise. Only valid when the **promote** command is invoked and `need_return` is true.

**role**

The name of the role currently selected. This will be empty unless the **role** command has been invoked.

**secure\_context**

Returns whether ACCEL is currently executing in a *secure context* — see "[Dangerous](#)" Functions.

**serverprocess\_id**

if running client/server, returns the current acserver's process id; if not running client/server, returns same as [process\\_id](#).

**sort1, sort2**

Hold two items to be compared by a user-defined sort function. Only valid when the `sortlist` function is invoked.

**symbname**

The symbolic name associated with a reserved part (i.e. a part that has been issued for edit). This variable is only valid when the **issue edit** or the **return unedit** commands are invoked.

**show\_obsolete**

Whether or not to show obsolete items in ACE browsers, i.e. whether or not the **Show Obsolete Items** option has been selected.

**timezone**

The name of the current timezone, e.g. "GMT Daylight Time".

**tmpdir**

The full path of the temporary directory used by *AllChange*.

**toolbar\_check**

Returns whether currently checking the (greying) conditions for the toolbar. This variable is only valid in ACCEL code in `menucond.ac`.

**topartname**

The full partname of a part being created. This variable is only valid when the **copy** or **rename** commands are being invoked.

**topathname**

The full path name of the object associated with a part being created. This variable is only valid when the **add**, **alter** (when changing the location), **copy** or **rename** commands are being invoked.

### **tree\_depth**

The current depth of the search through the parts tree. This variable is valid when the **report** or **find** or **baseline** commands are invoked.

### **true**

Returns *true*. Equivalent to any non-empty string in any truth context, but may be clearer when passing parameters, etc.

### **user**

The name of the current user.

### **vc\_client\_server**

Returns whether the client is currently running in VC client/server mode, i.e. the server is performing all operations on Version Control files on behalf of the client.

### **winsys**

The name of the windowing system under which the program is running. Current possible values are *windows*, *web* or '' (none). The value **web** will be used when accessing **AllChange** via ACC from the web browser interface.

### **workspace**

The name of the current workspace that a user is attached to.

## **Visual ACCEL**

### **About Visual ACCEL**

Visual ACCEL permits sites to design their own dialogs and hook them into ACE via ACCEL. (Note that this does not mean the appearance of dialogs/ windows built into ACE can be changed.)

A simple interface between ACCEL and the Front-end is supplied by the `prompt_...()` group of functions. These functions are limited to only being able to display predetermined controls, only being able to ask the user for one piece of information at a time, and not being visually very "pleasant". Visual ACCEL overcomes all of these restrictions.

Visual ACCEL provides support for displaying arbitrary (modal) dialogs from ACCEL. Help — either a simple dialog or a link into HTML Help — may be associated with these dialogs. Complex dialogs built into ACE, such as the Part List selector or ACCEL Condition Editor, are accessible. ACCEL user-defined variables are used for setting the initial contents of controls in the dialog before displaying it and retrieving the user's selections when the dialog is exited. It is also possible to respond to user interaction while the dialog is present.

A single DLL, `ACUIDLG.DLL`, is supplied with ACE to provide this functionality. It contains both the code required to communicate with ACCEL and a number of dialogs which are used by ACE. Dialogs are created/ modified directly in this file.

In order to create your own dialogs you will need a Windows Resource Editor. Microsoft's Visual C++ environment is suitable; other standalone resource editors may be available. Unfortunately, Visual Basic will not do the job as the dialogs it creates are interwoven with VB code. You will *not* need to write C code — nor will you need a C compiler or a linker — provided the controls and facilities you use are supported by the code supplied by Intasoft. If you do wish to use more advanced facilities, the source code we use is supplied and may be modified and recompiled as necessary.

Without any need for recompilation, the code as supplied by Intasoft allows for an unlimited number of user-defined modal dialogs, each using any combination of the following controls:

- Up to 10 checkboxes
- Up to 10 radiobuttons
- Up to 10 edit controls (single- or multi-line)
- Up to 10 text controls
- Up to 10 listbox controls (single or multiple selection)
- Up to 10 combobox controls (with or without edit control)
- Up to 10 date picker controls
- Up to 3 filelist controls
- Up to 3 partlist controls
- Up to 1 report formatlist controls
- Up to 1 condition editor controls
- Up to 3 directory list controls
- Up to 10 listview controls (with headings and icons)
- Up to 10 treeview controls (with icons)
- **OK, Cancel, Abort, Retry, Ignore, Yes, No** pushbuttons
- Up to 10 additional pushbutton controls
- Disabling of any of above controls
- Hiding of any of above controls
- "Callbacks" to respond to user interactions with certain controls
- Tooltips

This list is subject to change; see the comments in the source code file `acuidlg.c` for the latest support.

The steps required for using a user-defined dialog from ACE are as follows:

1. Use a resource editor to create/ amend dialogs directly in `ACUIDLG.DLL` (as stated above, VC++ at least allows you to do this by opening the DLL directly). Follow the rules detailed below when creating new dialogs and controls.
2. Decide where the dialog is to be invoked in **AllChange** (commands file, cycle file, report format file etc.).
3. Set ACCEL variables for the desired initial contents of the dialog's controls (see below).
4. Invoke the ACCEL function `do_dialog()` to display the dialog to the user, allow him to interact with the controls and press a pushbutton to exit the dialog.
5. Read ACCEL variables to discover the user's selections in the dialog (see below), and then act on them as required.

As supplied, `ACUIDLG.DLL` places requirements on user-defined dialogs as follows:

- Dialog IDs must be in the range 1000–1999.
- Checkbox IDs must be in the range 440–449.
- Radiobutton IDs must be in the range 460–469.
- Edit control IDs must be in the range 480–489.
- Text control IDs must be in the range 500–509.
- Listbox control IDs must be in the range 520–529.
- Combobox control IDs must be in the range 540–549.
- Filelist control IDs must be in the range 560–569. Each filelist control must have a corresponding edit control with id 1 greater than it (e.g. filelist id 560, edit id 561).
- Partlist control IDs must be in the range 570–579. Each partlist control must have a corresponding edit control with id 1 greater than it (e.g. partlist id 570, edit id 571).

- Formatlist control IDs must be in the range 580–589. Each formatlist control must have a corresponding edit control with id 1 greater than it (e.g. formatlist id 580, edit id 581).
- Condition editor control IDs must be in the range 590–599. Each condition editor control must have a corresponding edit control with id 1 greater than it (e.g. condeditor id 590, edit id 591).
- Directory list control IDs must be in the range 600–609. Each directory list control must have a corresponding edit control with id 1 greater than it (e.g. dirlist id 600, edit id 601).
- Listview control IDs must be in the range 610–619.
- Treeview control IDs must be in the range 620–629.
- Date picker control IDs must be in the range 630–639
- The following control IDs are used to exit the dialog, and should be associated with the specified pushbutton:
  1. **OK**
  2. **Cancel**
  3. **Abort**
  4. **Retry**
  5. **Ignore**
  6. **Yes**
  7. **No**
- The control ID 9 is reserved for invoking a special "help" system (pressing **F1** will do the same). If a pushbutton is created with this ID then pressing that button will cause a dialog whose ID is equal to the current dialog's ID + 1000 to be displayed. This dialog should contain only static text and an **OK** pushbutton which, when pressed, returns the user to the original dialog. So, for example, if you create a dialog whose ID is 1001 it can be accompanied by a corresponding "help" dialog by placing a **Help** pushbutton with ID 9 on the dialog and creating another dialog with help text and a single **OK** button and assigning the help dialog an ID of 2001. Alternatively, as described with [do dialog](#), the **Help** pushbutton may invoke the HTML Help system.
- Additional pushbutton control IDs must be in the range 10–19. Unlike pushbutton control IDs 1–7 these do not exit the dialog; they may be used to activate a callback event.

All controls of a single type (checkboxes, radiobuttons etc.) should be numbered contiguously (i.e. no gaps) starting from the lowest ID in their range. The tab order may be set as desired. Group boxes and other "furniture" may be added as desired. To actually place radiobuttons in a group remember that Windows requires the **Group** style to be set on the first radiobutton in a group and on the *next* control in tab order *after* the radiobutton group. Checkboxes and radiobuttons must have the **Auto** style set. Other styles may be set as desired.

The code allows for a double-click on an item in a listbox to select the item and then select the default button on the dialog, as in many applications. Any listbox whose ID is 520 (`IDC_LIST0`) and with the `LBS_NOTIFY` style (set by the **Notify** checkbox in the Resource editor) will exhibit this behaviour; this may be disabled by either assigning the listbox an ID other than 520 or removing the **Notify** style. A preferable method for accomplishing this — which also works for listview as well as listbox controls — is via the `ACCEL` variable `DialogDefaultList`, as described below.

Listview controls (multicolumn list boxes with headings) are intended to be used with their View mode set to **Report** (i.e. each item is displayed with multiple columns of information); at present sorting is not catered for, so **Sort** should be left on **None** and you may want to select **No sort header**.

Date picker controls can be used with either the **Short Date** or **Long Date** formats, not the **Time** format. By default, the Windows control does not allow no date to be selected. Setting the **Show None** style displays a checkbox which (when cleared) does allow this; if this style is set then the checkbox is cleared if

DatePickerIn[0-9] is empty, and DatePickerOut[0-9] is set to the empty string if the checkbox is cleared on exit.

The static text of the dialog title, checkbox, radiobutton, text and pushbutton controls, and help text, may all be altered when the dialog is displayed at run-time. This means that generic dialogs may be designed and then be used in different contexts.

To display a user-defined dialog, use this ACCEL function:

**do dialog** (*dllname, dlgnum, dlgproc, helpfile, helpstr*)

Invokes a modal user-defined dialog, and waits for the user to interact with it and exit.

On initialisation ACUIDlgProc looks for the following ACCEL user-defined variables to set the contents of the controls:

**CheckBoxesIn**

State of checkboxes. This is in the format of a space-separated string (ACCEL list) of ON or OFF words according as the corresponding checkbox is checked or not. Uninitialised controls are left unchecked.

**CheckBoxTextIn [0-9]**

Text of checkboxes. Use these to override the design-time text. Uninitialised controls retain the control's original design-time text.

**ComboItemsIn [0-9]**

Content of combobox controls. Set any of these ACCEL variables to the list of items for the combobox control. Uninitialised combobox controls are left empty.

**ComboSelectedIn [0-9]**

Initially selected item, if any, in corresponding ComboItemsIn [0-9].

**CondeditorArgs [0]**

List of arguments to be passed to ACCEL prompt\_condedit() when corresponding Condition button is pressed.

**CondeditorIn [0]**

Content of condedit controls. Set any of these ACCEL variables to the initial text for the condedit control. Uninitialised controls are left empty.

**ControlsDisabled**

State of controls. This is in the format of a space-separated string (ACCEL list) of the IDs of any of the controls to be disabled. Uninitialised controls are left enabled.

**ControlsHidden**

State of controls. This is in the format of a space-separated string (ACCEL list) of the IDs of any of the controls to be hidden. Uninitialised controls are left shown.

**ControlsTooltips**

Specifies any tooltips for use with the controls in the dialog. This should be set to a list of items, each of which is of the form:

```
<control-ID>^t<tooltip-text>
  where ^t is a tab character.
```

**DatePickerIn [0-9]**

The value for the date initially selected. This should be in standard Intasoft format (YYYY/MM/DD).

**DialogCallback**

Name of a user-defined ACCEL function to call in response to a user action. If uninitialised there is no callback.

**DialogCallbackControls**

Space-separated string (ACCEL list) of the IDs of any controls which are to activate the callback. If uninitialised no controls cause a callback.

**DialogDefaultList**

The ID of a listview (or listbox) control which — if an item in it is double-clicked on — is to be treated as though the dialog's default button had been clicked (after first selecting the item). If this is empty there is no default list control.

**DialogTitle**

Title of the dialog. Use this to override the design-time title. If uninitialised the dialog's original design-time title is retained.

**DirlistArgs [0-2]**

List of arguments to be passed to ACCEL `prompt_dirlist()` when corresponding Dirlist button is pressed.

**DirlistIn [0-2]**

Content of directory list controls. Set any of these ACCEL variables to the initial text for the directory list control. Uninitialised controls are left empty.

**EditCheckSpelling [0-9]**

Enables spell-checking for an edit control. Set to *true* to enable, or *false* to disable. The default value is *false*. Note that this cannot be used for an HTML control.

**EditHTMLEditor [0-9]**

Enables HTML editing for an edit control. Set to *true* to enable or *false* to disable. The default value is *false*. Note that spell-checking cannot be set on an edit control that is set to the HTML editor. (If both are set, the HTML editor option takes precedence.)

**EditTextIn [0-9]**

Content of edit controls. Set any of these ACCEL variables to the initial text for the edit control. Uninitialised controls are left empty.

**EditTextMaxLen [0-9]**

Limits the length (i.e. number of characters the user may enter) of corresponding `EditTextIn[0-9]`. If uninitialised the corresponding edit control is not limited.

**FilelistArgs [0-2]**

List of arguments to be passed to ACCEL `prompt_filelist()` when corresponding Filelist button is pressed.

**FilelistIn [0-2]**

Content of filelist controls. Set any of these ACCEL variables to the initial text for the filelist control. Uninitialised controls are left empty.

**FormatlistArgs [0]**

List of arguments to be passed to ACCEL `prompt_formatlist()` when corresponding Formatlist button is pressed.

**FormatlistIn [0]**

Content of formatlist controls. Set any of these ACCEL variables to the initial text for the formatlist control. Uninitialised controls are left empty.

**HelpText**

If using the internal help dialog, control ID 500 (`IDC_TEXT0`) is set to this text. So, to set the help text dynamically, create a static control with id 500 on the help dialog and set this variable to the desired help text. If uninitialised the dialog's original design-time text is retained.

**ListItemsIn [0-9]**

Content of listbox controls. Set any of these ACCEL variables to the list of items for the listbox control. Uninitialised listbox controls are left empty.

**ListSelectedIn [0-9]**

Initially selected item(s), if any, in corresponding `ListItemsIn[0-9]`.

**ListViewHeadingsIn [0-9]**

Text of listview controls column headings; use the tab character to separate columns. Listview

controls *must* have headings, and the number of columns in the heading determines the number of columns in the listview control.

#### **ListViewIconsIn[0-9]**

Icons to be used in listview controls. Set any of these ACCEL variables to the list (space separated string) of icon IDs to be used as item images. The small (16x16) version of an icon is used and so should be defined. You may add your own icon resources to supplement those Intasoft has supplied; we suggest you assign your own icons IDs > 200. Uninitialised listview icon lists cause the listview control to have no item images.

#### **ListViewItemsIn[0-9]**

Content of listview controls. Set any of these ACCEL variables to the list of items for the listview control; within each item use the tab character to separate columns (cf. `ListViewHeadingsIn[0-9]`). If the corresponding `ListViewIconsIn[0-9]` is not empty, the first column of each item is treated as the *index*, starting at 0, into the corresponding `ListViewIconsIn[0-9]` of the desired icon image. Note that `ListViewSelectedIn[0-9]` and `ListViewItemsOut[0-9]` *never* have this icon-index first column. Uninitialised listview controls are left empty.

#### **ListViewSelectedIn[0-9]**

Initially selected item(s), if any, in corresponding `ListViewItemsIn[0-9]`.

#### **PartlistArgs[0-2]**

List of arguments to be passed to ACCEL `prompt_partlist()` when corresponding Partlist button is pressed.

#### **PartlistIn[0-2]**

Content of partlist controls. Set any of these ACCEL variables to the initial text for the partlist control. Uninitialised controls are left empty.

#### **PushButtonTextIn[1-19]**

Text of pushbutton with corresponding ID. Use these to override the design-time text. Uninitialised controls retain the control's original design-time text.

#### **RadioButtonsIn**

State of radiobuttons. This is in the format of a space-separated string (ACCEL list) of `ON` or `OFF` words according as the corresponding radiobutton is selected or not. Uninitialised controls are left unselected.

#### **RadioButtonTextIn[0-9]**

Text of radiobuttons. Use these to override the design-time text. Uninitialised controls retain the control's original design-time text.

#### **TextIn[0-9]**

Content of text controls. Set any of these ACCEL variables to the initial text for the text control. Uninitialised controls are left empty.

#### **TextTruncate[0-9]**

Controls truncation of text controls whose text extends beyond the edge of the control. The value may be *path*, *end* or left empty. Specifying *end* causes any text that does not fit to be truncated with an ellipsis (...) on its right-hand edge. A value of *path* causes the beginning, and the right-most part of a file or part path to be preserved, with an ellipsis inserted before the last part. An empty value will truncate the right-hand edge with an ellipsis.

#### **TreeViewIconsIn[0-9]**

Icons to be used in treeview controls. Set any of these ACCEL variables to the list (space separated string) of icon IDs to be used as item images. The small (16x16) version of an icon is used and so should be defined. You may add your own icon resources to supplement those Intasoft has supplied; we suggest you assign your own icons IDs > 200. Uninitialised treeview icon lists cause the treeview control to have no item images.

**TreeViewItemsIn [0-9]**

Content of treeview controls. Set any of these ACCEL variables to the list of items for the treeview control; each item must look like:

*level*^t[*icon-index*^t[*icon-selected-index*^t]]*label*

where:

^t is the tab character ("^t" in ACCEL)

*level* is the desired level in the tree, the root level starting at 0

*icon-index* is the *index*, starting at 0, into the corresponding `TreeViewIconsIn [0-9]` of the desired icon image. *icon-index* (and its following ^t) is optional: if an item does not have it the first icon in the list will be used.

*icon-selected-index* is the *index*, starting at 0, into the corresponding `TreeViewIconsIn [0-9]` of the desired icon image *when the item is selected*; for example, when a folder in a treeview is selected it usually has an "open" image. *icon-selected-index* (and its following ^t) is optional, and may only be specified if an *icon-index* has been specified too: if an item does not have it the icon specified by *icon-index* will always be used for the item whether it is selected or not.

*label* is the desired label

Uninitialised treeview controls are left empty.

**TreeViewSelectedItemIn [0-9]**

Initially selected item, if any, in corresponding `TreeViewItemsIn [0-9]`; format is as `TreeViewItemOut [0-9]`.

`ACUIDlgProc` clears all of these input variables prior to exiting. This means that it is not necessary to reset them to empty before each new call to `do_dialog()`.

On exit `ACUIDlgProc` sets the following ACCEL user-defined variables to the selections/ contents of the controls:

**CheckBoxesOut**

State of checkboxes (cf. `CheckBoxesIn`).

**ComboItemOut [0-9]**

Selected item in combobox controls (cf. `ComboItemsIn [0-9]`).

**CondeditorOut [0]**

Condition entered in condition editor controls.

**DatePickerOut [0-9]**

The date selected in standard Intasoft format (YYYY/MM/DD).

**DialogCallbackCauser**

Set to the ID of the control which was activated to call the callback (valid only during a callback).

**DialogExit**

Set to the text of the exiting pushbutton (OK, Cancel etc.), with any & accelerator characters removed.

**DirlistOut [0-2]**

Selected items in directory list controls.

**EditTextOut [0-9]**

Content of edit controls (cf. `EditTextIn [0-9]`).

**FilelistOut [0-2]**

Selected items in filelist controls.

**FormatlistOut [0]**

Selected items in formatlist controls.

**ListItemsOut [0-9]**

Selected item(s) in listbox controls (cf. `ListItemsIn [0-9]`).

**ListViewItemsOut [0-9]**

Selected item(s) in listview controls (cf. `ListViewItemsIn [0-9]`). Returns the complete item(s), including tabs.

**PartlistOut [0-2]**

Selected items in partlist controls.

**RadioButtonsOut**

State of radiobuttons (cf. `RadioButtonsIn`).

**TreeViewItemOut [0-9]**

Selected item in treeview controls (cf. `TreeViewItemsIn [0-9]`), as full path in tree, with tabs separating item levels.

**Callbacks**

Visual ACCEL supports "callbacks" from dialogs. This is a very powerful facility allowing the content of a dialog to be altered in response to user interaction. It permits, for example, the enabling/ disabling of edit controls as the user selects checkboxes/ radiobuttons, changing the content of a list as the user selects an items in a combobox, doing something in response to clicking an arbitrary pushbutton, validating input when the user tries to **OK** the dialog, and so on — in short, the sort of thing done in the inbuilt dialogs in ACE and other standard Windows programs.

When `ACUIDlgProc` is called it looks to see whether variables `DialogCallback` and `DialogCallbackControls` are defined. If `DialogCallback` is set and the user performs one of the following actions on a control whose ID is among `DialogCallbackControls` the user-defined function named by `DialogCallback` is called (with no parameters):

- Checkbox/ radiobutton/ any kind of pushbutton: click (or keypress)
- Combobox/ treeview: click (or keypress) on an item
- Listbox/ listview: click (or keypress) on an item, or double-click on an item

On entry to the user-defined function all the ACCEL user-defined variables which are usually set on exiting the dialog (`CheckBoxesOut et al`, other than `DialogExit`) are set to reflect the current state of the dialog. In addition, all the variables which were read when `ACUIDlgProc` was invoked (`ComboItemsIn [0-9] et al`) retain their values, except that those relating to current selection (`CheckBoxesIn et al`) are updated to reflect the current state. The following variables are also set:

**DialogCallbackCauser** set to the ID of the control which was activated to call the callback.

**DialogCallbackEvent** set to a string describing the event which happened on the control to call the callback. Possible values are `click` (includes keypresses) or `dblclick`.

The function is then at liberty to examine the ACCEL output variables and as a consequence hide/ show or enable/ disable controls, change list/ combobox contents and so on by altering the ACCEL input variables read by `ACUIDlgProc`. If any variables are changed the function *must* return *true*, in which case the dialog will be repopulated in accordance with these values; if the function decides not to perform any alterations it should return *false*, in which case the dialog will remain unchanged. If you have a callback on a pushbutton which would normally exit the dialog (e.g. **OK**), or on the default dialog list specified via the `DialogDefaultList` variable, if the function returns *true* the dialog is *not* exited after all; this may be used to perform input validation before allowing the dialog to be exited. It is possible to display another Visual ACCEL dialog from this function, but you must then be careful to save/ restore any Visual ACCEL "In" variables affected by the second dialog prior to exiting the callback function and reentering the first dialog.

**Notes**

The supplied `ACUIDLG.DLL` includes sample dialogs. There is also a supplied `acuifunc.ac` file which defines functions that display these dialogs; it also illustrates callbacks. Inspect the code of this function

for Example of how to interact with Visual ACCEL. To test out the facilities try reading in this file (e.g. `read_functions('acuifunc.ac', '')`). Then from the **Info | Eval...** menu enter: `call(acuidlg)`. There are also many features in ACE which use Visual ACCEL — search the function definition files for calls to `do_dialog()`.

The source code, resource files, buildfiles etc. used for constructing `ACUIDLG.DLL` are supplied in the `ACUI` subdirectory of the **AllChange** system directory. If you find you need to provide more or different controls, or different event handling, you may modify and recompile the code. Obviously, however, upgrading will be simpler if you stick with the facilities as supplied.

With regard to upgrades, Intasoft may supply new versions of `ACUIDLG.DLL` over time with enhanced functionality in the code. Since you may wish to take advantage of such new code but retain existing dialogs (we will do our best to ensure that new code still works with existing user-defined dialogs) you will need to copy all your dialogs from your DLL and paste them into the new `ACUIDLG.DLL`: your resource editor should allow this. To avoid this, you may find it desirable initially to copy `ACUIDLG.DLL` to another filename, modify dialogs in this file and pass its name to the ACCEL `do_dialog()` function.

NOTE: we have discovered that Windows 95/98 (but not NT) does *not* permit the direct editing of resources in DLLs. To create/ modify dialogs under 95/98 you should instead load/ save dialogs as `.RC` files from VC++ and then use the resource compiler and the linker to regenerate a DLL. We suggest you always work on a copy of `ACUIDLG.DLL` as above. You may wish to work this way on all platforms anyway.

### Wizard Dialogs

Visual ACCEL has been used to implement "wizard" dialogs. These are a sequence of dialogs which present the user with a series of simple questions in order to accomplish some task at the end. The user can move backward as well as forward through the dialogs, so previous answers can be reviewed or altered. If appropriate, answers may be saved for reuse in a future session, and help can be provided for each dialog.

Functions which achieve this using Visual ACCEL are supplied in the file `wizfunc.ac`. They are used to implement the . . . **Report Wizard** dialogs which guide the end user through the process of running a report, and for the **Column Report Wizard** dialogs which allow the Administrator to create columnar report format files. Use these models to create your own wizards.

# AllChange Project Settings Editor

## About AllChange Project Settings Editor

The **AllChange** project settings editor (ACPROJSET) allows registry entries to be created/ modified.

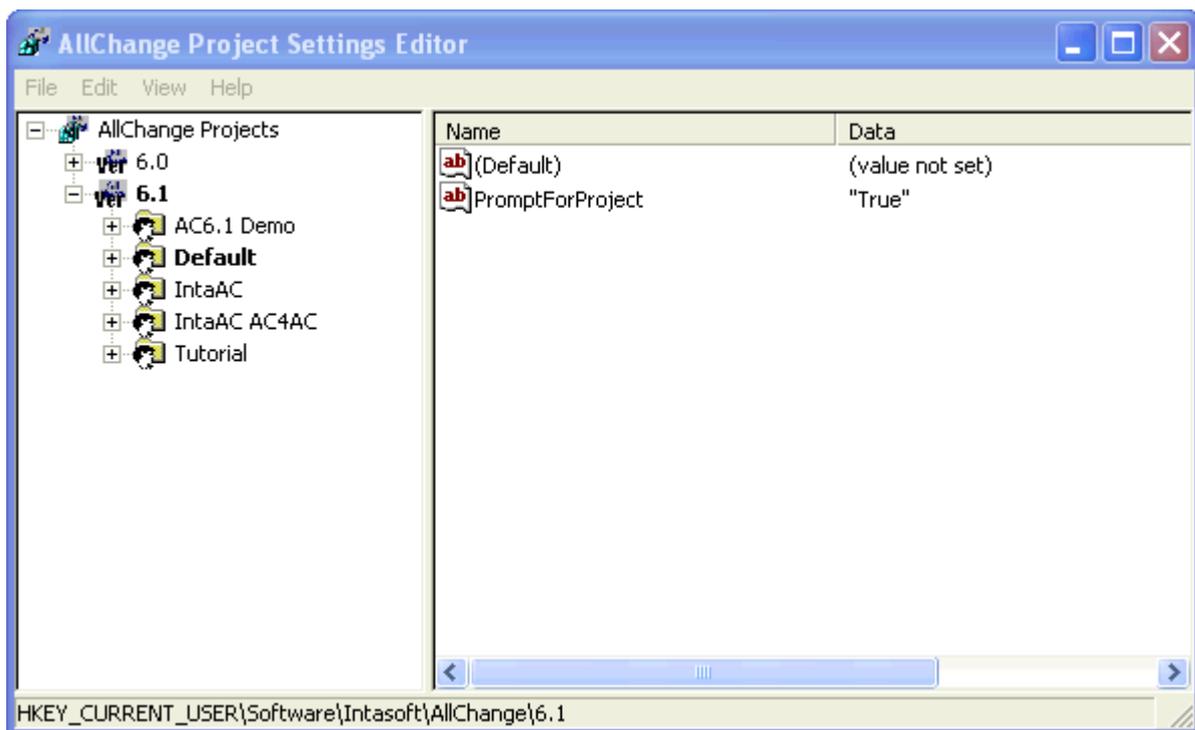
**AllChange** programs store information about each project in the registry. In most cases it is not necessary to modify the registry using the projects settings editor as the installation tool, ACEand ACCONFIG will create and modify the registry as necessary for you.

However, there may be times when it is necessary to examine/ change the registry yourself (e.g. if you are not using project definitions as supported by ACCONFIG) and ACPROJSET provides a tool for doing this.

ACPROJSET will provide you with access to all registry entries for all **AllChange** projects (for each **All-Change** version), but will not allow you access to other parts of the registry.

A shortcut to ACPROJSET will only be created in the Intasoft group for administrators, end users are not intended to use ACPROJSET.

On entering ACPROJSET the main window will show the registry entries the subkey for each version of **AllChange** that exists in your registry.



Within each **AllChange** version there will be subkeys for each **AllChange** project and within those various subkeys for different sets of settings.

In addition at the **AllChange Projects** top level there is an entry called **CurVer** which determines which is the current **AllChange** version in use.

For each **AllChange** version there is an entry **ACDIR** which determines where the system directory is for that particular version.

By default ACPROJSET will show the settings in **HKEY\_CURRENT\_USER** which is where most settings are stored.

However for the server machine, if you are running client/server, the settings are stored in **HKEY\_LOCAL\_MACHINE**. These server settings may be accessed by selecting **View | Local Settings**.

Other **View** options include:

**Read Only:** if selected then the registry may only be viewed and not modified

**Status Bar:** if selected the status bar is shown

**Refresh:** causes the registry to be re-read

For details of the common registry entries used by **AllChange** see [Windows Registry Settings](#).

## Modifying the Registry

The registry entries displayed may be modified using various options on the **Edit** menu or the context sensitive menus.

**Modify:** allows a key value to be modified

**New:** allow a new **AllChange** version, project, key or value to be added to the registry

**Copy:** allows an **AllChange** version, project, key or value to be copied. For **AllChange** version, project and key a new entry is made named as **Copy of oldname**. For values the entry is copied to the clipboard and may be pasted in at a later date.

**Paste:** allows a copied registry *value* to be pasted into the current open key.

**Delete:** allows the currently selected item to be deleted.

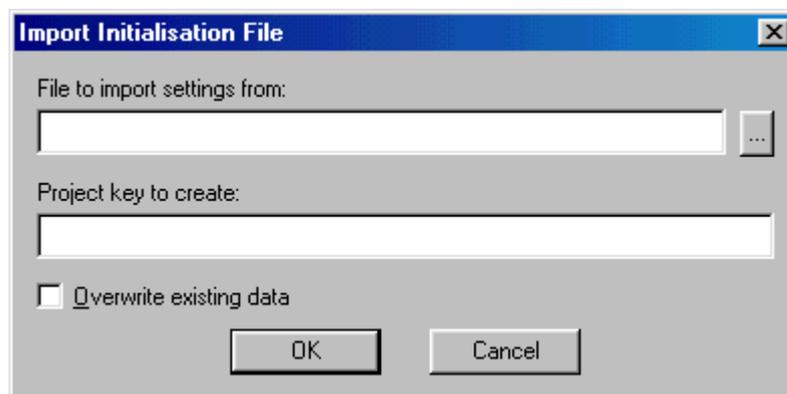
**Rename:** allows the currently selected item to be renamed.

**Set as Default Project:** allows the default project to be changed. This will be shown *ticked* when the current default project is selected and *unticked* otherwise.

## Importing "ini" Files

If you have a previous version of **AllChange** which used *intasoftini* files you may wish to create registry settings based on existing intasoftini files.

The **File | Import Ini File** option will do this for you.



Select the intasoftini file to be imported from, and specify the name of the new project key that the settings are to be imported into.

A new project entry will be created with values according to those that existed in the intasoftini file.

# Appendix A - Command Definitions in Depth

## Command Definitions in Depth

The command definitions file, `commands.ac`, defines the entry conditions and the actions used in the **AllChange** commands.

A sample configuration for the command definitions is supplied with the system. Many functions that it calls are supplied in various supporting files including:

```
cmdfunc.ac for functions directly affecting commands
vcfunc.ac for version control tool interface functions
utility.ac for general useful functions
```

Depending on your intended usage of **AllChange** these may be used "as are", or they may require some tailoring to meet particular site requirements. A large amount of configuring may be achieved simply by setting entries in the **Configuration Options**; `ACCONFIG` may be used to do this.

The requirements and suggested usage of the command definitions are given below. Which database(s), if any, has a current record is mentioned; any fields of that record should be accessed by a plain reference (not a qualified one). (Virtually all field references for both conditions and actions will be plain references to the current record.) Any special ACCEL variables which are valid for each entry are also listed.

When writing actions it is important that any commands which access a component's VC file, or other file/directory which may reside on the server, use the appropriate ACCEL function where one is available. These include: `control()`, `putaway()`, `takeout()`, `takeout_for_edit()`, `vcerase()`, `vcinfo()`, `vcprep()`, `vcscan()`, plus a number of functions with `vcfile/dir` or `serverfile/dir` in their name.

A command which is not one of the in-built commands that are automatically monitorable may be made monitorable by adding a call to the ACCEL function `check_monitor()` at the end of the actions section — see [Creating Monitor Events](#).

There is one entry per **AllChange** command in the `commands` file, except for **history**. In addition there are special (pseudo command) entries for `ignoreprotecttext`, `ignoreprotecttextbaseline`, `ignoreprotecttextpart`, `interpret`, `monitor_action`, `readbaseline`, `readbaselinevote`, `readcr`, `readcrvote`, `readinstance`, `readmonitor`, `readpart`, `readpartvote`, `readbaselinestatuslog`, `readcrstatuslog`, `readpartstatuslog`, `showitemshiddenmessage`, `voteinitiated`, `votedecision`, `voteblocked`.

In general the following rules apply:

1. For entry conditions the current record is the record *prior* to any changes. For the actions the current record is the record *after* any changes.
2. The entry conditions may *not* perform any processing which may change the database.
3. For the actions, any changes to the database caused by the command will not have been flushed to the database. The new data may be referenced via the current record(s) set up for the command actions. Any invocation of `interpret()` (or `db_tran_flush()`) will cause a flush of all outstanding transactions to the database.
4. If an action *alters* a VC file in any way it should be followed immediately by a `db_tran_flush_no_backout()`. This means that the transaction thus far cannot be backed out and ensures consistency between the state of the database and the VC files.

There now follows an alphabetical list of all the **AllChange** commands and the pseudo commands together with the requirements on their entry conditions and actions. This should be read in conjunction with inspecting the actual entries in the supplied `commands.ac` file.

### addbaselinevote

**ACCEL variables:** none

The current record for the condition and for the action is the vote record.

There are no particular requirements for the **addbaselinevote** entry.

### **addcrvote**

**ACCEL variables:** none

The current record for the condition and for the action is the vote record.

There are no particular requirements for the **addcrvote** entry.

### **addpartvote**

**ACCEL variables:** none

The current record for the condition and for the action is the vote record.

There are no particular requirements for the **addpartvote** entry.

### **alter**

**ACCELvariables:** `frompathname`, `topathname`

The alter action only has particular requirements if the location is altered.

The current record for the condition and for the action is the part being altered.

For parts of type subsystem the command actions should create the directory (and VC sub-directory if appropriate) for the new location if it is different from the old location. The directory may not be renamed since some other part may refer to the original location and so must still exist.

If the part is of type component , the command actions should rename the existing file represented by the part to the new filename represented by the part if the new location is different from the old and the file for the new location does not already exist.

The ACCEL `frompathname` variable is set to the full path for the old part location and the ACCEL `topathname` variable is set to the full path for the new part location if the location field is altered.

ACEinvokes this entry with an argument of `text=memory` or `text=memory2` whenever a Part text has been changed in ACE. This argument must never be used in any other circumstance.

### **alterbaseline**

**ACCELvariables:** none

The condition is invoked once for the baseline, and then for each detail altered. The action is invoked once for the baseline, and for each detail altered. The actions for each detail are performed before the action for the baseline.

The current record for the condition and for the action is the baseline record. When the condition/ action is invoked for the baseline, then the `bl_partname` will be empty (i.e. `bl_partname == ''`). For the condition and action for the detail, the current part record will be the part corresponding to the detail if it exists.

The condition should prevent the baseline from being altered if it is locked. There are no special requirements of the action.

ACEinvokes this entry with an argument of `text=memory` whenever a baseline text has been changed in ACE. This argument must never be used in any other circumstance.

### **alterbaselinevote**

**ACCEL variables:** none

The current record for the condition and for the action is the vote record.

There are no particular requirements for the **alterbaselinevote** entry.

## altercr

**ACCEL variables:** none

The current record for the condition and for the action is the CR record.

There are no particular requirements for the **altercr** entry.

ACE invokes this entry with an argument of `text=memory` whenever a CR text has been changed in ACE. This argument must never be used in any other circumstance.

## altercrvote

**ACCEL variables:** none

The current record for the condition and for the action is the vote record.

There are no particular requirements for the **altercrvote** entry.

## alterinstance

The **alterinstance** entry has no specific requirements. The current record for the condition and the action is the instance together with the version and the component related to this instance.

## alterissue

**ACCEL variables:** `symbname`

The current record for the condition is the issue/check-out record. The current records for the action are the issue/check-out record and the corresponding part record.

The condition should prevent a check out read only being changed to a check out for edit if the part is already checked out for edit anywhere, and ensures the user is attached to the workspace of the current check-out/issue.

If changing a check out read only to a check out for edit then there is a current part and a valid `symbname` variable during the action; the **Takeout** tool is called to retrieve the appropriate version of the file represented by the part, overwriting the existing workfile. The action moves the workfile from one directory to another if the workspace to which the part is checked out is being changed.

If the user to whom a part is checked out for edit is altered the subsequent return from edit will be performed by a different user from the original check out for edit. The requirements for this are discussed under the entry for the **return** command.

## alterpartvote

**ACCEL variables:** none

The current record for the condition and for the action is the vote record.

There are no particular requirements for the **alterpartvote** entry.

## alterversion

**ACCEL variables:** `comment`, `symbname`

The current record for the condition and for the action is the part record.

The action part looks to see if the `no_ver` flag is being added or deleted. If it is being added — which is what normally happens in the course of a check out for edit — then the version is taken out for edit (creating a branch if the new version is the first on a branch) as in the **issue** command (the `symbname` variable holds the new symbolic name). If it is being deleted — which normally happens during a **return** from edit — then the version is put away as in the **return** command (the `comment` variable holds the comment). This behaviour allows sites to use **newversion** to create new versions instead of the usual **issue/return** mechanism; however, this is not recommended unless explicitly desired, hence the permission requirement.

The current working directory is changed to the work directory corresponding to the part for the duration of the actions.

ACE invokes this entry with an argument of `text=memory` or `text=memory2` whenever a version text has been changed in ACE. This argument must never be used in any other circumstance.

## assigncr

**ACCEL variables:** none

The current record for the condition and for the action is the CR record.

The actions should send information to the user assigned the CR informing him of the assignment.

## attachws

**ACCEL variables:** none

The entry conditions should check that the user is allowed to attach to the workspace.

The current record for both the condition and action is the workspace to be attached to.

There are no particular requirements for the actions.

## baseline

**ACCEL variables:** none

The condition is invoked once for the baseline, and then for each detail added. The action is invoked once for the baseline, and then for each detail added.

The current record for the condition and for the action is the baseline record. There will also be a statuslog record if one is to be created.

When the condition/ action is invoked for the baseline, then the `bl_partname` will be empty (i.e. `bl_partname == ''`). For the condition and action for the detail, the current part record will be the part corresponding to the detail if it exists.

## build

**ACCEL variables:** `generate_depends`, `object`

The condition and action are invoked for each target which is to be built, as specified on the command line. The ACCEL variable `object` holds the name of the target.

The actions are responsible for invoking the **Build** tool explicitly. They are invoked twice for each target being built: the first time the ACCEL variable `generate_depends` will be true and **Build** must be called to generate the `depends.ac` file (using the undocumented `-z` flag as shown below); the second time `generate_depends` will be false and **Build** must be called to actually perform the build (using the `-b` flag to tell **Build** to use BTs). Any special options used in the `-options` argument are passed on to the **Build** tool.

## command

**ACCEL variables:** none

There are no special requirements for the command condition or action. The action may cause the ACE Output window to be up-fronted on completion of the command.

## copy

**ACCEL variables:** `frompathname`, `topartname`, `topathname`

The condition is invoked just once for the command: the current record is the parent of top-level new record. The actions are invoked for each part copied; the current record will be the part being copied, the ACCEL `topartname` will contain the full partname of the new part, the ACCEL `topathname` variable

will contain the full path for the location of the new part, and the ACCEL `frompathname` variable will contain the full path for the location of the part being copied.

For parts of type **component** the actions should copy the file corresponding to the part copied from, to the **location** for the part copied to.

For parts of type **subsystem** the actions should create a directory (and a VC sub-directory if appropriate) for the **location** of the part copied to.

## cwp

**ACCEL variables:** none

The current record for the condition and for the action is the part record. There are no special requirements for the conditions or actions.

## delete

**ACCEL variables:** `frompathname`

The condition and actions are invoked for each part to be deleted, starting with the bottom of the tree.

The part being deleted is the current record of the parts database for both the condition and the action.

The entry condition may, for example, check that the part is not locked (i.e. is not contained in any database), and should check that it is not currently checked out to anyone.

The actions should delete the files associated with component type parts and *may* delete the directory associated with subsystem type parts if the directory is not referred to by any other part. The ACCEL variable `frompathname` is set to the full path for the part's location.

## deletebaseline

**ACCEL variables:** none

The condition is invoked once for the baseline, and then for each detail deleted. The action is invoked for each detail deleted and then once for the baseline.

The current record for the condition and for the action is the baseline record. When the condition/ action is invoked for the baseline, then the `bl_partname` will be empty (i.e. `bl_partname == ''`). For the condition and action for the detail, the current part record will be the part corresponding to the detail if it exists.

It is suggested that the entry condition for deleting a baseline tests that the baseline is not locked.

## deletebaselinevote

**ACCEL variables:** none

The current record for the condition and for the action is the vote record.

There are no particular requirements for the **deletebaselinevote** entry.

## deletecr

**ACCEL variables:** none

The **deletecr** entry is used when deleting a CR.

The current record for the condition and for the action is the CR record.

There are no particular requirements for the **deletecr** entry.

## deleteissue

**ACCEL variables:** none

The **deleteissue** entry is used when deleting an issue/check-out record via the `deleteissue` command.

The current record for the condition and for the action is the issue/check-out record.

Note that no workspace is attached to or current directory changed during **deleteissue**, so there should be no attempt to do any file-based actions.

### **deletecrvote**

**ACCEL variables:** none

The current record for the condition and for the action is the vote record.

There are no particular requirements for the **deletecrvote** entry.

### **deleteinstance**

The **deleteinstance** entry has no specific requirements. The current record for the condition and the action is the instance together with the version and the component related to this instance.

### **deletemonitor**

**ACCEL variables:** none

The deletemonitor conditions/ actions have no special requirements.

The current record for the condition and for the action is the monitor record.

### **deletpartvote**

**ACCEL variables:** none

The current record for the condition and for the action is the vote record.

There are no particular requirements for the **deletpartvote** entry.

### **deleteversion**

**ACCEL variables:** none

The **deleteversion** entry is used when deleting an individual version; note that it is *not* used when deleting a whole component(which will include all its versions).

The current record for the condition and for the action is the part record.

The entry condition may, for example, check that the version is not locked (i.e. is not contained in any base-line), and should check that it is not currently checked out to anyone.

The actions should delete the corresponding version from the version control file using the **Vcerase** tool. The version to be deleted is defined by the symbolic name field. It is important that the version only be deleted from the version control file if it genuinely exists, hence the checks on the flags.

### **edit**

**ACCEL variables:** none

The **edit** actions should invoke the required editor with the work filename for the part as a parameter.

The current record for the condition and for the action is the part record.

The editor command could be taken from an ACCEL variable which might be set up in the initialisation file.

### **editcrtext**

**ACCEL variables:** none

This entry is only invoked for an explicit **editcrtext** command; it is not used when editing a CR text in ACE.

The current record for the condition and for the action is the CR record.

The condition has no special requirements. It could be made to make the same checks as **putcrtext** if the `-view` argument is *not* used (since it will end up doing a **putcrtext**); however, this would mean that users would have to be very careful to use the `-view` argument.

The actions must do **getcrtext** to copy the CR text from the database into an external file and then invoke the editor on the resulting file. This should be followed by **putcrtext** to store the result back in the database or the copy of the CR text should simply be deleted as appropriate. The text is to be deleted if the `-view` argument was explicitly used; as an optimisation, even if this argument was not used the text is also deleted if the modification time of the file has not changed since before invoking the editor, i.e. no changes were made.

## eval

**ACCEL variables:** none

There are no special requirements for **eval**

## find

**ACCEL variables:** none

The condition and action are invoked just once for the command. There is no current record.

There are no special requirements for the condition or the action. The action may cause the ACE Output window to be up-fronted on completion of the command.

## getcrtext

**ACCEL variables:** none

This entry is only invoked for an explicit **getcrtext** command; it is not used when editing a CR text in ACE.

There are no special requirements for the conditions or actions.

The CR whose text is being extracted is the current record of the CRs database.

## ignoreprotecttext, ignoreprotecttextbaseline, ignoreprotecttextpart

ACCEL variables: none

These are special (pseudo command) entries and do not correspond to a command.

The condition is invoked just before displaying the text in a viewer window and determines whether the text is editable. An error will cause the text to be *protected*, no error will make the text editable.

There is no action.

## interpret

**ACCEL variables:** `command_name`, `command_line`

If this entry exists its actions section (only) is executed at the beginning of any new interpret prior to the command being executed. The ACCEL variable `command_name` holds the name of the command. Interpret is executed in response to the ACCEL `interpret` function *and* in response to a top-level user action. This entry can be used to do something once prior to executing a command on a number of items.

Example: suppose the user chooses to change the status of a number of parts and you want to prompt for a single comment to apply across all parts. Since the `status` entry is called once for each part in turn you need some way to distinguish the first occasion and only prompt then. This can be achieved by setting a `firsttime` variable in the `interpret` actions and testing and clearing this in the relevant status actions in the `cycles.ac` file, e.g.:

```
interpret
action
  setvar(firsttime, true);
end
```

```
Editing
action
```

```
if var(firsttime) then
    setvar(reason, prompt(...));
    setvar(firsttime, '');
endif;
interpret('issue -edit -comment "'.var(reason)." '.pa_part);
end
```

## issue

**ACCEL variables:** `symbname`

The `issue` command is the underlying `AllChange` command which performs a check out.

The version history of parts is maintained using the supplied version control tools and the link between the part versions and the versions in the VC files is defined by the symbolic name field for each part version, which must be assigned to the appropriate version in the VC file. It is therefore important that the command actions perform the required operations to ensure that this link is maintained.

The current record for the condition is the part to be checked out. The current record for the actions is the part checked out plus the new check-out record. The `symbnameACCEL` variable is the new symbolic name for the reserved version set up by a check out for edit (`issue` command with `-edit` argument).

The condition could be used to prevent parallel development by checking the part is not already checked out for edit.

The current working directory is changed to the work directory corresponding to the part for the duration of the actions.

If the part is not being checked out for edit then the actions should retrieve a *read-only* copy of the appropriate version of the part into the current workspace.

If the part is being checked out for edit then the actions should retrieve a *writable* copy of the appropriate version into the current workspace.

The actions should use the `ACCEL takeout` function to call the **Takeout** tool to retrieve the appropriate version of the file represented by the part. The version to be retrieved is defined by the symbolic name field of the part.

`make_bt_file` should be called to create a BT file for the version checked out; the BT file is deleted beforehand in case anything goes wrong.

## list

**ACCEL variables:** none

There are no special requirements for the condition or action. The action may cause the ACE Output window to be up-fronted on completion of the command.

## monitor

**ACCEL variables:** none

The current record for the condition and for the action is the monitor record.

There are no special requirements for the condition or action.

## monitor\_action

**ACCEL variables:** `monitoritem`

This is a special entry and does not correspond to a command.

The actions are invoked each time an event occurs on an item which is monitored for that event. The condition is not used.

The actions should inform the holder of the monitor that the event has occurred on the item. The ACCEL variable `monitoritem` is also available, containing the name of the item for which the monitor has been invoked.

The monitor triggered is the current record of the monitors database.

### newcr

**ACCEL variables:** none

The **newcr** entry has no specific requirements. The current record is the new CR for both the condition and the action. There will also be a statuslog record during the action if one is to be created.

The actions may store the created CR number in a user-defined variable (this may be used elsewhere).

### newinstance

The **newinstance** entry has no specific requirements. The current record for the condition and the action is the instance together with the version and the component related to this instance.

### newversion

**ACCEL variables:** `comment`

The current record for the condition is the part together with the version which is the predecessor of the new version to be created. The current record for the action is the part together with the new version. There will also be a statuslog record during the action if one is to be created.

Sites may use **newversion** to create new versions instead of the usual **check-out/check-in** mechanism; however, this is not recommended unless explicitly desired. The `-noinitial` argument should only be used on the first version. Flags should not be settable on the first version.

If it is the first version that has been created then the actions must perform the VC actions for creating the first version in the corresponding VC file; the `-noinitial` argument determines whether this will be empty or not. If it is a subsequent version the actions must perform the VC actions for putting away the version. The `comment` variable holds the comment. The current working directory is changed to the work directory corresponding to the part for the duration of the actions.

### promote

**ACCEL variables:** `object`, `pool_object`, `need_promote`, `need_return`, `return_from_edit`

The condition and action are invoked for each object which is to be promoted and/ or returned. The ACCEL variable `object` holds the name of the object.

If the object is a part then the current record for the condition and the action is the part.

The entry condition is responsible for invoking the **Build** tool explicitly *if a check is required that an object is built up-to-date before it may be promoted.*

The actions should perform all the necessary functions to implement the promote command, together with any **returns** required.

If the object is a part and needs to be returned (indicated by the `need_return` ACCEL variable), then the actions should return it (from edit if necessary). Note that the `-keep` argument to **return** is used so that the workfile does not get removed so that it can be promoted to the pool after it has been **returned**.

If the object is to be promoted (indicated by the `need_promote` ACCEL variable) then the object (together with its BT) should be copied to the pool and then removed (unless it is still checked out). If the `-edit` argument was used then the object's BT should be updated prior to copying it to the pool.

### putcrtext

**ACCEL variables:** none

This entry is only invoked for an explicit **putcrtext** command; it is not used when editing a CR text in ACE.

The **putcrtxt** entry has no special requirements. However, it is worth mentioning here that the CR text should be a text file; specifically, it must not contain a byte of 0 (ASCII NUL).

The current record for the condition and for the action is the CR record.

## quit

**ACCEL variables:** none

The action (but not condition) is executed just before exiting ACC/ACE.

The action should be used to perform any desired clean up operations.

## readbaseline, readbaselinevote, readcr, readcrvote, readinstance, readmonitor, readpart, readpartvote, readbaselinestatuslog, readcrstatuslog, read-partstatuslog

ACCEL variables: none

These are special (pseudo command) entries and do not correspond to a command.

The entry conditions can be used to prevent read-access to baselines, CRs, parts or instances, or baseline, part, instance or CR fields based on other fields, roles, etc. This is provided as a refinement to the command-access settings for the pseudo-commands.

Care should be taken when adding or modifying code in these conditions, as any field references will themselves be subject to these conditions, and this code will be re-entered. Fields not available for command-access in `acconfig` are not tested here.

To prevent read-access to an item, or a field, the condition should error. Note though that the text of the error will not normally be seen by the user, except when viewing an item for which any access is denied, where the error text is displayed on the viewer instead of the usual tab-pages.

There is no action.

### Examples:

To hide the text of CRs in the status of released, use the following code in the 'entrycond' section of the 'readcr' pseudo-command (note that the first three lines prevent the condition from being re-entered when referencing `cr_status` in the condition):

```
if getarg('status') then
return ''; # stop the reference to cr_status evaluating this condition
endif;
if cr_status == 'released' and (getarg('text') or getarg('text_raw')) then
error_map("You may not view the text of released #Cr_s#");
endif;
```

To hide the subsystem '/protected', and all of its children from users other than administrators, add the following code to the 'entrycond' section of the 'readpart' pseudo-command:

```
if not is_admin_user and (pa_part == '/protected' or relative_part('/-
protected', pa_part) != '') then
error_map("Only Admin users can view protected #part_s#");
endif;
```

## release

**ACCEL variables:** `pool_object`, `object`

The condition and actions are invoked for each object released.

The condition may implement any desired access restrictions.

The actions should copy the released object from the pool to the release directory using the ACCEL `pool_object` and `object` variables respectively.

## rename

**ACCEL variables:** `frompathname`, `topartname`, `topathname`

The current record for both the condition and the action is the part being renamed.

The entry conditions should check that the part is not currently **issued** (checked out), since if the part is renamed whilst a check-out is still outstanding, then an attempt to check in the part checked out will fail since the original part no longer exists.

The **rename** actions should rename the file corresponding to the part renamed to that corresponding to the new part. The ACCEL variable `topartname` holds the new name for the part, the ACCEL variable `topathname` holds the new location for the part, and the ACCEL variable `frompathname` holds the old location for the part.

## renamebaseline

The current record for both the condition and the action is the baseline being renamed.

The condition should prevent the baseline from being renamed if it is locked. There are no special requirements of the action.

## report

**ACCEL variables:** none

There are no special requirements for the condition.

The action may cause the ACE Output window to be up-fronted on completion of the command.

## return

**ACCEL variables:** `comment`

The condition and the action have both the part returned and the issue/check-out record as the current records.

If the part was checked out optimistically then the current version is the temporary optimistic version (i.e. `<part>;%<workspace>`) in the condition, and the final 'real' version (i.e. `<part>;<version>`) in the action. Also, if the invoker of the command has the version as the current record, then an unqualified `pa_part` reference will deliver the optimistic version before the **return**, and the real version after the **return**. For a pessimistic check-out, the version id does not change during the **return** command, as the correct final version is calculated on check-out.

The condition may be used to enforce that the user who did the check-out is the user who does the check-in. Alternatively it might allow any user with an appropriate role to do the return, or it might not place any restrictions at all on who can return. Furthermore it may wish to test whether the `-edit` argument has been used and have different requirements accordingly.

The condition also ensures that if the `-baseline` argument has been used to specify a baseline against which the return is being made then that baseline must exist and not be locked.

The current working directory is changed to the work directory corresponding to the part for the duration of the actions.

The actions for a simple **return** (i.e. not with the `-edit` or `-unedit` arguments) should simply remove the working copy of the file represented by the part that was created by the corresponding **issue**.

If the `-edit` argument was used the actions should return the file from the workspace back to the location for the part.

The working copy of the file should be stored under version control using the ACCEL `putaway` function to call the **Putaway** tool. The ACCEL `comment` variable should be used as the comment associated with the new version. This will contain any comment specified as an argument to the **return** if one was specified or

the comment associated with the check-out if there is one or the symbolic name for the part. The version to be **Putaway** should be the symbolic name associated with the part.

If a version is returned from edit, normally the *user* field of the version will be set to hold the name of the user doing the return: this may differ from the user who did the check-out for edit. There is an entry in the configuration options (see [Configuration Options](#)) to alter this behaviour: if the **On return from edit log issuer** is enabled then the name kept in the version record will remain that of the person who checked out for edit regardless of who checks in.

If the `-unedit` argument was used then the actions should cancel the outstanding edit if there is one. The actions should use the **Control** tool to cancel the edit and the working copy of the file represented by the part should be deleted.

The BT file for the checked out version should be deleted.

For all of the above cases if the `-keep` argument was used then the working copy of the part (and its BT) should be left in the workspace (i.e. it should not be removed). This may be implemented by using the `-k` argument to the **Putaway** tool when this is being used and simply not deleting the workfile in other cases.

## role

**ACCEL variables:** none

There are no special requirement for the condition or actions.

## set

**ACCEL variables:** none

There are no particular requirements of the **set** entry. However, it is recommended that strict access permissions are enforced for switching on/ off "dangerous" flags, such as *actions*.

## status

**ACCELvariables:** *newstatus,oldstatus*

The current record for the condition and for the action is the part record. There will also be a status log record for the action if one is to be created.

The condition may be used to implement access controls for the command in general. Access controls based on the status change requested may be implemented in the life-cycle definition.

The actions are performed *before* any actions from the life-cycle.

## statusbaseline

**ACCELvariables:** *newstatus,oldstatus*

The current record for the condition and for the action is the baseline record. There will also be a status log record for the action if one is to be created.

The condition may be used to implement access controls for the command in general. Access controls based on the status change requested may be implemented in the life-cycle definition.

If it is not desired that a locked baseline's status can be changed, then the condition should check that the baseline is not locked.

The actions are performed *before* any actions from the life-cycle.

## statuscr

**ACCELvariables:** *newstatus,oldstatus*

The current record for the condition and for the action is the CR record. There will also be a status log record for the action if one is to be created.

The entry for **statuscr** may be used to implement access controls for the command in general. Access controls based on the status change requested may be implemented in the life-cycle definition.

The actions are performed *before* any actions from the life-cycle.

## use

**ACCEL variables:** none

There are no special requirements for the condition or the action.

The current record for the condition is the parent of the uses type part. The current record for the actions is the new uses part.

## voteblocked

**ACCEL variables:** none

This is a special entry and does not correspond to a command.

The actions are invoked when a decision is blocked on a vote. The condition is invoked just before the action.

The actions should send the blocked vote email..

The vote definition and the item voted on are the current records.

## votedecision

**ACCEL variables:** none

This is a special entry and does not correspond to a command.

The actions are invoked when a decision can be reached on a vote. The condition is invoked just before the action.

The actions should send the decision vote email.

The vote definition and the item voted on are the current records.

## voteinitiated

**ACCEL variables:** none

This is a special entry and does not correspond to a command.

The actions are invoked when a new vote is opened (by moving an item to a status which has a vote defined). The condition is invoked just before the action.

The actions should send the vote Initial email..

The vote definition and the item voted on are the current records.

## voteypassnext

**ACCEL variables:** none

The current record for the condition and the action is the vote definition and the item being voted on.

The actions should send the vote Initial email.

## Appendix B - Integration with Version Control Tools

### Appendix B — Integration with Version Control Tools

**AllChange** makes use of underlying facilities provided by the version control tools supplied with the system, to perform the actual storage and retrieval of versions of files and build tools to call up the operating systems required to produce derived files. The VC Tools are documented in their own manual, *AllChange VC Tools Manual*, should this ever be needed.

This Appendix details a number of miscellaneous factors to be borne in mind for the smooth interaction between **AllChange** and these tools. It also describes how to "import" the information contained in existing VC files into the **AllChange** system so that **AllChange** can be used to manage the versions from now on.

### The VC System

The following points should be borne in mind when setting up and using the **AllChange** system:

- The VC tools are used to store and retrieve actual versions of files and for configuration building.
- Versions in VC files should only be manipulated through **AllChange** commands, *not* by the tools directly. In particular, you must not create or erase versions or alter the assignment of symbolic names other than through **AllChange**. Failure to adhere to this rule may result in conflicts between what **AllChange** believes to be in the VC files and what is actually in them, leading to undefined behaviour.
- If the **Putaway when no differences** flag is set on a VC file to not putaway (this is the default) it is vital that the `ACCELputaway` function is used to invoke **Putaway** so that when no version is created in the VC file the **AllChange** system can detect this. In general, all operations which call on VC Tools should go through a dedicated ACCEL function.

### VC Keywords

VC keywords may be placed in workfiles, as described in the *AllChangeVC Tools Manual*. This allows version history information to be embedded in workfiles — and possibly other files generated from them — for informational purposes. Their use is encouraged.

There are some special VC keywords, `$ACpart$`, `$ACversion$`, `$ACheader$`, `$AClog$` and `$ACident$` which may be used to connect an **AllChange** component with its VC file and an **AllChange** version with its VC version respectively.

**Takeout** substitutes the corresponding **AllChange** part name and version id where appropriate when generating the workfile. By placing these strings in workfiles it will be possible to see which **AllChange** part/version a workfile came from after future **return/Putaways** and **issue/Takeouts**. These keywords will also be extracted by **Vcident** so it may be possible to identify the **AllChange** parts/versions used to build an executable, as described in the *AllChangeVC Tools Manual*. These keywords are also understood by **Vcrep**.

The **AllChange** partname substitution string must be stored in the VC file whenever it is created, copied or renamed. To set the **AllChange** part for a file use a `-p` argument to **Control** or **Putaway** (via the [control](#) and [putaway](#) ACCEL functions) thus:

```
control ("-pallchange-partname", vcfile)
putaway ("-pallchange-partname", part, keep)
```

The **AllChange** version substitution string must be stored in the VC file against a version whenever it is created. To set the **AllChange** version for a file use a `-v` argument:

```
control ("-vallchange-version",, vcfile)
putaway ("-vallchange-version", part, keep)
```

These arguments may be used together, e.g.

```
control("-pallchange-partname -vallchange-version",, vcfile)
```

Note that the supplied ACCEL code which needs to invoke the VC Tools built in functions such as `control`, `takeout` and `putaway`, are invoked via a corresponding ACCEL function defined in `vcfunc.ac` called **VCControl**, **VCTakeout** and **VCPutaway** respectively

The supplied command definitions (`commands.ac` file) use these arguments where appropriate.

The comment substitution for the `$AClog$` keyword is taken from the comment stored in a version arbitrary field named `Comment` if there is one, otherwise the CRs Solved by the version. It is possible to customise the comment part of the log using the `-kc` argument to `takeout` where it is invoked (via `VCTakeout`) from the `commands.ac` file:

```
call (VCTakeout, "-kccomment.....",part, workfile)
```

The *comment* could, for example be taken from a version arbitrary field such as:

```
call (VCTakeout, quote_os("-kc".unquote(pa_varb1))." otherargs",part, workfile)
```

`$ACident$` keyword, is recognised by `Takeout` and its value is taken from the `-ki` argument to **Takeout**. This may be used to stamp other arbitrary information about a part into an extracted workfile. For example it may be desirable to stamp the workfile with the CRs which authorised the changes to the version:

- calls to `VCTakeout` in `commands.ac` should be modified to include:

```
quote_os('-kiAllChange CRs = '.crssolved(pa_part, ''))
```

in the arguments

- If the `$ACident$` keyword is put into the file then after Check Out, the workfile will contain something like:

```
quote_os('$ACident: AllChange CRs = CR00101$')
```

Using these keywords in `takeout()/takeout_for_edit()` alone will cause `Putaway` to detect differences in an otherwise unchanged workfile (since it cannot know what was specified at `Takeout` time), and hence create a version unnecessarily. Consequently, `Putaway` accepts corresponding `-tki<substitution>` and `-tkc<comment>` arguments, and uses these during its internal comparison against the predecessor version. It is recommended that these are used to specify to `Putaway` the same substitutions as were used during `Takeout`; remember, if these reference values taken from a version this must be the version which was taken out, i.e. the predecessor of the version being put away.

## Appendix C - System Configuration

### Appendix C — System Configuration

This Appendix describes various means by which the **AllChange** system may be configured to meet requirements. It includes a summary of the various system files used by **AllChange**. [Operating System Specifics](#) should also be consulted for information on operating system specific issues.

## System Files

**AllChange** system files are files used to control the behaviour of **AllChange** tools. Depending on the purpose of the individual system file, **AllChange** will search for it in one or more of the current directory, home directory, workspace directory, **AllChange**project directory or **AllChange**system directory (where the system directory includes any relevant project template directory).

The current directory (or current working directory) is set by the appropriate normal operating system command (e.g. `cd`). From within ACC or ACE it is set by attaching to a workspace and by changing the current working part.

The home directory is a directory private to the individual user. This may be a special directory for **AllChange** use only as specified by the `ACHOME` environment/ registry entry or may be the user's real home directory – [Operating System Specifics](#) explains what it is set to under each operating system.

The workspace directory is a directory private to a particular workspace — see [Workspaces](#).

An **AllChange** project directory contains system files used by **AllChange** programs. By setting up different project directories different system files may be maintained for use by different projects. System files which may vary from project to project are generally sought in the **AllChange** project directory (if there is one).

To indicate that he wishes to work on a particular project the user must set an environment variable (or an entry in the `.ini` file) called `ACPROJDIR` to the directory containing the project's system files. The project directory may also be set (or overridden) from the command line when invoking an **AllChange** program by using a `-EACPROJDIR=...` argument. There is no default project: if `ACPROJDIR` is not set there is no project and **AllChange** will look only in the **AllChange** system directory for system files.

The **AllChange** system directory contains system files used by **AllChange** programs. System files which do not vary from project to project are generally sought only in the **AllChange** system directory; system files which may vary between projects may be sought in the system directory if not found in the project directory.

There is no default **AllChange** system directory: the user *must* set an environment variable (or an entry in the `.ini` file) called `ACDIR` to the directory containing the **AllChange** system files.

The **AllChange** system uses a number of files to hold a variety of information about the system. Below is a list of the names of the files used, their location, a brief description of their function and an indication of where they are described in more detail if appropriate.

In most cases normal **AllChange** users need not be aware of the contents of these files, or even of their existence. The System Administrator, however, must ensure these files are properly installed and maintained in line with the site's requirements. The list is divided into two: those files the Administrator may wish to tailor in order to affect the system configuration, and those which are simply required for the system to run or are generated internally.

The following system files may be tailored to affect system configuration:

### `ac.ini`

[project then system directory] If this file is present it contains **AllChange** commands which are executed on entry to ACC and ACE before interacting with the user — see [The Startup File](#).

### `acefunc.ac`

[project then system directory] This file contains ACCEL functions used by ACE.

### `acjobs.acx`

[project directory] This file contains the definitions of any scheduled jobs.

### `acjobtimes.acx`

[project directory] This file contains last run time and next run time for any scheduled jobs.

**acspwd.ac**

[system directory] This file contains user names and passwords for server based login to AllChange — see [AllChange Logon](#). It is comprised of lines of the form:

*username [password]*

**ACUIDLG.DLL**

[executable directory] This file contains the dialogs which can be displayed from Visual ACCEL— see [Visual ACCEL](#). Dialogs can be directly edited in this file.

**acdwwcs.dll**

[executable directory] This file contains the Adobe Dreamweaver interface functions — see [Integration with Dreamweaver](#)

**admusers.ac**

[project directory] This file contains a list (one per line) of those users who may be administrators of the **AllChange** system. Certain operations can only be performed by administrative users — see [User Registration](#).

**archfunc.ac**

[project then system directory] This file contains ACCEL functions to support the archiving of CRs, baselines and parts.

**blexport.rep**

[project then system directory] This file is used to support the archiving of baselines.

**branches.ac**

[project directory] This file contains the names of branches that may be used. The format of each line is:

*branchname top-part*

**browser.ac**

[project then system directory] This file defines the layout and contents of ACE browsers — see [Browsers](#). The file is composed of blank-line-separated sections; the format of each section is:

*listname database*  
`[. +][?]heading width [realval=expression] expression`  
 |

Fields are separated by whitespace. *Listname* identifies the list in ACE that is being specified. The names that appear in the supplied file should not be modified and must all exist. *Database* specifies the **AllChange** database being displayed in the list. This value should not be changed from the supplied setting. *Heading* is the text of the column heading. *Width* is the default column width; a negative column width means left truncate. *Expression* is an ACCEL expression returning the display value for the column. If *realval=* is present this is used for the sort order. Fields starting with a . are not shown by default; those starting with a + are vital and cannot be hidden; those starting with a ? are arbitrary fields and have the site-defined arbitrary field name as the heading.

**build.ini**

[current then project then system directory] File containing initial rules, macros etc. read in by **Build** on initialisation.

**build.tpl**

[current then project then system directory] File containing include file patterns, initial buildfile template etc. read in by **Autobuild**.

**chkinoutfunc.ac**

[project then system directory] This file contains ACCEL functions to implement the **Check In** and **Check Out** functions

**classes.ac**

[project then system directory] This file details the classes known to **AllChange**. It must be present

for the class and life-cycle aspects of **AllChange** to function — see [Classes](#). The format of each line is:

```
[+]class-name class-type {attrib=attribute} [cycle-name [cr-num-format [cr-template]]]
```

Fields are separated by whitespace. The default class is specified by preceding the class name with a +. *Attribute* can be: "Require CR".

#### **cmdfunc.ac**

[project then system directory] This file provides ACCEL support functions for `commands.ac` and other support files.

#### **commands.ac**

[project then system directory] This file defines the entry conditions and actions for nearly all **AllChange** commands. This is the key file to the functioning of most aspects of the **AllChange** system — see [Command Definitions](#).

#### **condedit.ac**

[project then system directory] This file determines what items will be offered in the ACCEL condition editor in ACE — see [ACCEL Condition Editor](#). The file is composed of blank-line-separated sections for each defined context. The format for each section is:

```
context
field-name field data-type display-name
op|cont
ACCEL expression returning list of values for operator
end
[?]field-name field data-type display-name [index-name]
rhs
ACCEL expression returning list of values for field
end
function-name function
p1|p2|p3 descriptive string
ACCEL expression returning list of values for parameter
end
```

Fields are separated by whitespace. Each *context* may have one or more of the above subsections.

The `Default` context is used to define the operators used for each data type; it should not be changed. In order to allow English-type phrases for operators, the `Default` context allows 3 special field entries, all with `mapops` as the *field-name* and the following as the *data-type*:

#### **map\_real**

A list of actual ACCEL operators. A value in this list may contain `%l` and `%r` to indicate where to insert the *lhs* and *rhs* of the operator into any function calls. If neither is found it will be treated as a conventional operator.

#### **map\_display**

A list of English-type phrases that will be displayed to the end user when selecting an operator. The items in this list must be in the same order as those in the `map_real` list.

#### **map\_multiselect**

A list of any operators (as they appear in the `map_display` list) which take a list of possible values as their *rhs*.

Arbitrary fields may be specified with `?` before the *field-name*: this will cause the defined field name in the `fldnames.ac` definitions to be presented. The ACCEL function `arbitrary_field_values()` may be used to return the list of valid values of the *rhs*. Any value in the resulting list of values may start with a `+`: this will then be used as the default value selection.

The field *data-type* is optional but if specified may be one of:

- date
- list
- longstring
- part
- string
- truth
- user

The *data-type* is used to determine which operators in the `Default` context should be used.

The *descriptive string* for functions is displayed above the list for that parameter when the function is selected.

#### **config.ac**

[project then system directory] This file contains miscellaneous entries affecting certain aspects of the overall behaviour of the **AllChange** system — see [Configuration Options](#). The format of each line is:

*name=value*

Valid built in *names* in entries are:

```

AcconfigNoBackup
AcconfigNoOverwrite
AcconfigSaveWritable
AllowCheckInNoWorkspace
AllowKeepReadOnlyChanges
AllowNoVC
AllowOptimisticLocking
AllowReadPermissions
AllowUnlicensedUsers
AllowUserBranches
AutoUpdatePools
BinarySuffixes
BlineTargetReleaseFields
CRsForBaselineIncludeLinks
CRTargetReleaseFields
CheckLockoutTimes
ComponentLifeCycle
CRCycle
TextSections
CycleViewerEnable
CycleViewerHideInvalid
CycleViewerShowBadRole
DataSetStoreNonExistents
DataSetThresholdAccessTime
DataSetThresholdExpirationTime
DataSetThresholdSize
DataSetThresholdSlots
DateFormat
DbCheckLock
DbServerSetSavepoint
Derived_Deliverable_Patterns
DOORSWorkspace
DOSfilenames
EditableMailMessages
EnableFTPDeployment
EnableFTPPools
EnableFTPWorkspaces

```

EnableSpellChecking  
ErrorLogging  
FileAttachmentsAsLinks  
FormatMailAsHTML  
HideBaselineStatusLog  
HideCRStatusLog  
HidePartStatusLog  
ImportFromEmailAllowVote  
ImportFromEmailMailbox  
ImportFromEmailMailSystem  
ImportFromEmailNotForAllChange  
ImportFromEmailPop3Server  
ImportFromEmailReply  
InstanceDigits  
JointEdit  
KeywordsUseLocalTime  
LocationBelowWorkspace  
FindPathWorkspaceTryCurrent  
AskSelectWorkspace  
MailSystem  
MSProjectDateBaselineFinishFields  
MSProjectDateBaselineStartFields  
MSProjectNotesFromCRText  
MSProjectResourceFromCR  
MSWordReqTitles  
NoAutoBaselineStatusLog  
NoAutoCRStatusLog  
NoAutoPartStatusLog  
NoBaselineBaselinesAffected  
NoBaselineFilesAffected  
NoBaselineVotes  
NoBTs  
NoCRBaselinesAffected  
NoCRCRsAffected  
NoCRFilesAffected  
NoCRPartsAffected  
NoCRVotes  
NoDataSet  
NoDbClientLockBuffer  
NoDbSelect  
NoFileDoActions  
NoInstances  
NoKeywordSubsts  
NoMonitors  
NoPartPartsAffected  
NoPartVotes  
NoReadLinkedRecords  
PutawayKeywordSubstsErrorBinary  
PutawayNoDiffsForce  
PutawayStoreFiletime  
ReplaceSolved  
RequireCRAffected  
ReturnEditLogIssuer  
SendMailFromUser  
SendVoteMailFromUser  
ReviewCycles  
ShowProjectUsersOnly  
SMTPServer

```

SubsystemLifeCycle
TakeoutKeywordSubstsWarnBinary
TextFilesCompatible
UsesDoActions
ValidForChangeCRsAssigned
VCClientServer
VCErrLogFile
VCFileNamesCompatible
VCLogInfo
VersionDigits
VCCache
VCCacheFileMinSize
VCCacheRootDir
WebInterfaceURL
WebInterfaceAttachmentDir
WebInterfaceTempDir
WordStampDoc

```

*Value* depends on the entry; where this is a truth-type, `True` and `False` are used.

#### **cr.tpl**

[project then system directory] Default new CR text template file for CRs whose class does not specify one explicitly — see [Classes](#).

#### **crexport.rep**

[project then system directory] This file is used to support the archiving of CRs.

#### **crsforbln.ac**

[project then system directory] This file contains ACCEL functions to implement the **CRs for Baseline** function

#### **crwizrep.tpl**

[project then system directory] This is the template for report format files generated for columnar reports which are exported to Excel or Word — see [Columnar Reports in Excel/Word](#).

#### **crxl\*.xls**

[system directory] These files are used to display charts of CR information in Excel.

#### **cycfunc.ac**

[project then system directory] This file provides the ACCEL functions for the life cycle predefined conditions and actions

#### **cycles.ac**

[project then system directory] This file defines all the life-cycles known to the **AllChange** system. It must be present for the life-cycle and status aspects of **AllChange** to work — see [Life-cycles](#). The file is composed of blank-line-separated sections for each cycle. The format for each section is:

```

cycle-name cycle-type
status-name {status-progressions} {attrib=attribute} {flags}
entrycond|exitcond|entryaction|exitaction
ACCEL expressions
end
status-name {status-progressions} {attrib=attribute} {flags}
.
.
.

```

Note that, within each status entry, the `entrycond`, `exitcond`, `entryaction` and `exitaction` lines *must* be indented (with whitespace) from the left margin. Fields are separated by whitespace. *Status-progressions* is a comma-separated list of other statuses; use `.` (dot) if no progressions are permitted. *Attribute* can be one of: "CheckOut", "CheckIn", "Valid for Change",

"Release", "MSProject Start", "MSProject Bline Start" "Under Development" "Accepted", "Release Testing". *Flags* may be one or more of: newver, open, closed.

**diffsfunc.ac**

[project then system directory] This file contains ACCEL functions to support comparison and merge facilities in ACE.

**doorfunc.ac**

[project then system directory] This file provides ACCEL support functions for the DOORS integration — see [Integration with Telelogic DOORS](#).

**dragdropfunc.ac**

[project then system directory] This file contains ACCEL functions to support drag and drop functionality in ACE.

**dmwvfunc.ac**

[project then system directory] This file provides ACCEL support functions for the Dreamweaver integration — see [Integration with Dreamweaver](#).

**fldnames.ac**

[project then system directory] This file contains mappings from real field names to site specific field names — see [Field Name Definitions](#). The file consists of lines of the form:

```
db-prefix arbnun newname displayname {attributes}
{value-list}
```

Fields are separated by whitespace. *Db-prefix* is the (two letter) database prefix used in ACCEL to identify the database of which the field is a member — see [ACCEL Field References](#). *Newname* is the new name for the field. *Displayname* is the display name for the field. *Attributes* may be one or more of:

```
entry-
type=combo|fillin|readonly|multiselect|singleselect|accelselect
datatype=truth|string|long-string|list|date|date/time|
numeric|user|user-list
editable|noteditable
compulsory|optional x
class=classname
helpstring=helpstring
```

*Value-list* is either a list of values (one per line) or an ACCEL expression preceded by a & (ampersand) character. The *value-list*, if present, should be terminated by a blank line.

**fldtabs.ac**

[project then system directory] This file contains information as to which subtabs arbitrary fields should be displayed on and the order in which they are displayed. The file consists of sections of the form:

```
database tabname {classes}
{fieldname columnspan}
```

Fields are separated by whitespace. The *database* may be one of baseline, cr, part, version. *fieldname* is the **New Name** for the arbitrary field.

**ftpfunc.ac**

[project then system directory] This file contains functions to support FTP functionality — see [Integrated FTP support](#).

**ftppool.ac**

[project directory] This file contains mappings from local pools to remote directory names for FTP support — see [Integrated FTP support](#). The file consists of lines of the form:

```
poolname host host-os remote-path pool
```

Fields are separated by whitespace.

**ftpuser.ac**

[system directory] This file contains mappings from local usernames to remote usernames for FTP support — see [Integrated FTP support](#). The file consists of lines of the form:

```
username host remote-username
```

Fields are separated by whitespace.

**ftpwspc.ac**

[project directory] This file contains mappings from local workspaces to remote directory names for FTP support — see [Integrated FTP support](#). The file consists of lines of the form:

```
workspacename host host-os remote-path workspace|deploy
```

Fields are separated by whitespace.

**getver.ac**

[project then system directory] This file contains ACCEL functions to implement the Get Version and Baseline to Directory functions

**groups.ac**

[project and system directory] Used to hold group definitions — see [User Groups](#). The file consists of lines of the form:

```
group-name mailname=group-mailname {members}
```

Fields are separated by whitespace. *Members* is a comma-separated list of usernames. To allow for individual entries containing large numbers of usernames, entries may be continued across multiple lines by placing a \ (backslash) character as the very last character on a line; leading indentation on subsequent lines is skipped.

**htmlfunc.ac**

[project then system directory] This file contains functions to support HTML (Web) functionality — see [AllChange Web Browser Access](#).

**includes.ac**

[project then system directory] This defines the ACCEL function definition files known to **AllChange** and how/ whether they are used — see [Function Definition Files](#). The file consists of lines of the form:

```
filename in-use compulsory [comment]
```

Fields are separated by whitespace. *In-use* and *compulsory* may be true or false.

**intaweb.ini** [same directory as `accgibin.pl`] This is the ".ini" used by ACC when using the web browser interface to **AllChange**.

**lockout.ac**

[project directory] Used to lock users out of **AllChange** see [Lockout Times](#). The file consists of lines of the form:

```
[+|-][starttime endtime] message
```

Fields are separated by whitespace. Lines starting with a - (minus) cause a warning to be issued; lines starting with a + (plus) cause a fatal error to be issued and the user is immediately logged off from **AllChange**. If a line does not start with either of these characters it is treated as though it started with a +.

**mailfunc.ac**

[project then system directory] This file contains ACCEL functions to support email submission of CRs (only) — see [Email Submission of CRs](#).

**mailmap.ac**

[system directory] This file may be required under Windows depending on the mail system being used — see [Integration with Mail](#). The file consists of lines of the form:

```
logonname mailname
```

Fields are separated by whitespace. *Logonname* and *mailname* should be quoted where necessary.

**menucond.ac**

[project then system directory] This file provides facilities for configuring the user interface — see [Menu Item/ Toolbar Availability](#). The file consists of lines of the form:

```
action-name[ (parameter) ] availability helpstring  
[condition]
```

Fields are separated by whitespace. *Availability* may be one of: Never, Admin, FullOnly, Always. *Condition*, if present, should be terminated by a blank line.

**menuitem.ac**

[project then system directory] This file defines the contents and actions for the menu items be available from the ACE menus — see [Menu Items](#). The file consists of lines of the form:

```
menu-name menu-type [attrib=attribute] [towindow=destination] item-name action[ (parameter  
) ]  
[condition]
```

Fields are separated by whitespace. *Menu-type* may be: Menu, Cascade, Context or Drag. *Attribute* may be default, rename, delete, properties, control or shift. *Destination* names the destination window/ list; it should only be used when *menu-type* is Drag. *Item-name* should include any & (ampersand) characters required to indicate mnemonics, and . . . characters to indicate that a dialog will be shown, if appropriate. *Condition*, if present, should be terminated by a blank line.

**mprojfunc.ac**

[project then system directory] This file contains functions to support integration with Microsoft Project functionality — see [Integration with Microsoft Project](#).

**nomenmap.ac**

[project then system directory] This file contains nomenclature mappings between Intasoft standard terms and site specific terminology. The file consists of lines of the form:

```
intasoft-name new-name [p=plural-form] [a=apostrophised-form]
```

The plural and apostrophised forms are optional.

**paexport.rep**

[project then system directory] This file is used to support the archiving of parts.

```
local-path remote-path
```

Fields are separated by whitespace.

**pawdstmp.rep**

[project then system directory] This report is used during Word document stamping — see [Integration with Microsoft Word](#).

**pools.ac**

[project directory] This file details the pools known to **AllChange**. It must be present for the pool aspects of **AllChange** to function — see [Pools](#). The file consists of lines of the form:

```
pool-name pool-directory [server-pool-directory]
```

Fields are separated by whitespace.

**predefs.acx**

[project then system directory] This file contains the definition of the life cycle predefined conditions and actions that are available and their parameters. This file is in XML format.

**project.dic**

[project directory] This is the project defined dictionary, used by the spell-checking operations.

**projects.ac**

[system directory] This file contains the definitions of the **AllChange** projects which are available — see [AllChange Projects](#). The file consists of lines of the form:

```
[project name]
Comment=comment
ACPROJDIR=project directory
TEMPLATE=template
ACSQSERVER=SQL Server/instance
ACSQLPROJDB=Name of the SQL database
ACSqldbUser=SQL Server Authentication database user. Default use Windows Authentication if empty/not specified
ACSqldbPassword=SQL Server Authentication database user password. Stored encrypted
ACSqldbEncrypt=Set to True to enforce connection encryption. Default False
ACSERVER=server name
ACSERVERPROJDIR=server project directory
ACPORT=port number
UserAccess=[Everyone | Include | Exclude | None]
Users=user1 user2 ...
```

```
[project name]
```

```
...
```

**projfunc.ac**

[project directory] This file may contain ACCEL functions to be used by a project — see [Function Definition Files](#).

**projtmpl.ac**

[system directory] This file contains the definitions of the **AllChange** project templates which are available. The file consists of lines of the form:

```
[template name]
Location=project template directory
Description=Description of project template
Base=name of base template
AllowNew=True/False
```

The *project template directory* is the name of the directory containing the configuration files for the project template in the projtmpl subdirectory of the system directory.

The Base specifies which template this is a subset of/derived from. Any files which are not found in this project template directory will be next searched for in the Base template directory, thus allowing a hierarchy of templates to be used.

The *Intasoft Standard* template is the top level template equating to the system directory.

Certain files in template directories (excluding Intasoft Standard) may be named <configuration file name>\_EX.ac. This is an extension file to the normal <configuration file name>.ac file containing entries which are in addition to the normal file, or where the entry already exists, replacement for the normal file entry. \_EX files are only applicable if found in a directory before the full file is found.

Files which may have \_EX files include:

```
includes.ac
menucond.ac
menuitem.ac
classes.ac
predefs.acx
config.ac
cycles.ac
```

**register.ac**

[workspace directory] Each workspace may contain this file. It contains information pertaining to what pools and versions are used by the workspace — see [Register Definition Files](#). Ordinary **All-Change** users may wish to edit these files. Its format is:

```
{pool-names}
registration-entry
```

|

**report.sum**

[same directory as report format files] A file created by generating report summaries. Every report format file must have an entry in this file in order for it to appear on a **Report Formats** dialog — see [Report Formats](#). The file consists of lines of the form:

```
filename disabled=disabled access=access category=category description
{usernames}
```

Fields are separated by whitespace. *Disabled* may be `true` or `false`. *User-access* may be one of: `everyone`, `admin`, `named`. *Category* may be one of: `Brief`, `Detail`, `Function`, `Graph`, `HTML`, `Integration`, `List`, `Other`, `Status Log`, `Word Wizard`, `Word/Excel`. *Usernames* may appear, one per line and indented, if *user-access* was `named`.

**reportfunc.ac**

[project then system directory] This file contains ACCEL functions used by supplied reports

**rfc.tpl**

[project then system directory] Default new CR text template file for CRs whose class is `rfc` — see [Classes](#).

**rhafunc.ac**

[project then system directory] This file provides ACCEL support functions for the Rhapsody integration — see [Integration with Rhapsody](#).

**rolemap.ac**

[project then system directory] This file details the roles required for performing each **AllChange** command — see [Command Access](#). The file consists of lines of the form:

```
command [override] [class={classes}] [option[!]=option] {permissions}
status_entry [override] cycle=cycle status=status [option[!]=option]
{permissions}
```

Fields are separated by whitespace. *Command* may be either an **AllChange** command or an ACCEL function; alternatively it may be `status_entry`. *Classes* is a comma-separated list of classes. If present, `option=` means if *option* is used; `option!=` means if it has *not* been used. *Permissions* is a comma-separated list of roles and/ or ACCEL field references which resolve to a username (e.g. `cr_assignee`); a value of `(Any)` means any user may perform the command; a value of `(None)` means that no-one may perform the command.

**roles.ac**

[project then system directory] This file details the roles known to **AllChange**. It must be present for the access permission aspects of **AllChange** to function — see [Role Definitions](#). The file consists of lines of the form:

```
part role {users/groups}
```

Fields are separated by whitespace. *Users/groups* is a comma-separated list of usernames and/ or group names. To allow for individual entries containing large numbers of user/ group names, entries may be continued across multiple lines by placing a `\` (backslash) character as the very last character on a line; leading indentation on subsequent lines is skipped.

**sccifunc.ac**

[project then system directory] This file provides ACCEL support functions for the MCSCCI — see [Development Tool Integrations](#).

**shexfunc.ac**

[project then system directory] This file provides ACCEL support functions for the Explorer interface — see [Integration with Windows Explorer](#). It is also used by many other function files.

**shortcuts.acx**

[project then system directory] This file provides the default shortcut and folder list settings for ACE.

**svnfunc.ac**

[project then system directory] This file provides accel support functions for the facility to import parts from a subversion repository.

**task.tpl**

[project then system directory] Default new CR text template file for CRs whose class is `task` — see [Classes](#).

**toolbar.ac**

[project then system directory] This file determines the appearance and functionality of the toolbar in ACE. This file is laid out in 3 sections (in any order). Each section is terminated by a blank line.

1. Toolbar definitions — defines which buttons appear in each toolbar and under what conditions the toolbar is displayed. It consists of subsections of the form:

```
TOOLBARS
[?]toolbar-name condition (to end of line)
{button-names}
```

Fields are separated by whitespace. The `?` is optional and indicates that the toolbar should only appear if the user is a *full* user; i.e. the toolbar will not be displayed for CR-only users. *Condition* gives the ACCEL condition for when to display the toolbar. *Button-names* may be spread across multiple (indented) lines. A *button-name* of `space` produces a gap.

2. Category definitions — defines which buttons appear in each category on the **Insert Toolbar Item** dialog. It consists of subsections of the form:

```
CATEGORIES
category-name
{button-names}
```

*Button-names* may be spread across multiple (indented) lines.

3. Button definitions — defines each button, its descriptive text, quick help, button text, function etc. It consists of subsections of the form:

```
BUTTONS
button-name button-text descriptive-text
database function quickhelp condition
database function quickhelp condition
|
```

Fields are separated by whitespace.

**Button-name** is the name given in `toolbar.ids`.

**Button-text** is the name to appear on the button if **Show button text** is switched on.

**Descriptive-text** is the text to appear in the **Insert Toolbar Item** dialog.

**Database** is the database corresponding to the upfront window. This may be one of:

**Any** Any window.

**File** File browser.

**Part** Part browser or viewer.

**Issue** Issue(Check-out) browser or viewer.

**CR** CR browser or viewer.

**Monitor** Monitor browser.

**Baseline** Baseline browser or viewer.

**Workspace** Workspace browser.

**Role** Role browser.

**Function** is the function to use if the button is pressed. This may be one of:

**Exit** exit **AllChange**.

**ShowDialog (dialog)** show specified dialog. This may be one of: Add\_Part, Add\_CR, Add\_Baseline, Monitor, Alter\_Part, Alter\_CR, Alter\_Baseline, Alter\_Issue, Status\_Part, Status\_CR, Status\_Baseline, Report\_Part, Report\_CR, Report\_Monitor, Report\_Baseline, Report\_Issue, Report\_Workspace, Report\_Role, Report\_Pool, Report\_Cycle, Report\_Class, Report\_Command, List\_Part, List\_CR, List\_Monitor, List\_Baseline, List\_Issue, List\_Workspace, List\_Role, List\_Pool, List\_Cycle, List\_Class, List\_Command, EditList\_CR, EditList\_Baseline, Issue, Return, Promote, Build, Release, Eval, Set, OsCommand, AllChangeCommand, Rename, Copy, Use, Find, Assign\_CR.

**ShowWindow (window)** show specified window. This may be one of: File\_Browser, Part\_Browser, Part\_Viewer, Issue\_Browser, Issue\_Viewer, Baseline\_Browser, Baseline\_Viewer, CR\_Browser, CR\_Viewer, Role\_Browser, Workspace\_Browser, Monitor\_Browser.

**DeletePart** delete currently selected part.

**DeleteCR** delete currently selected CR.

**DeleteBaseline** delete currently selected baseline.

**DeleteMonitor** delete currently selected monitor.

**EditPart** edit currently selected part.

**EditCR** edit currently selected CR.

**ShowStatusLog** display status log dialog.

**ReadItem (-1)** Read previous viewer item.

**ReadItem (0)** Read current viewer item.

**ReadItem (+1)** Read next viewer item.

**UpdateItem** Update current Item from viewer.

**CallUserFunc (function)** Call specified user function.

**Quickhelp** Text of quick help (yellow tabs). Quote if more than one word.

**Condition** Condition to show button (greyed if false).

The name of each toolbar button is defined in a further file called `toolbar.ids`. To find which name applies to a picture, go into ACE, bring up the **Insert Toolbar Item** dialog and select **All Items**. The name of each item is show in brackets in the description field. Normally this file should not be altered. However, it is possible to add a site-specific button into ACE by changing the name of an unused button in `toolbar.ids` and adding a corresponding new entry into `toolbar.ac` to perform

the desired action. The actual button pictures may be altered using a tool which edits resources within an executable, e.g. Microsoft Visual Studio.

**toolbar.ids**

[project then system directory] This file in combination with `toolbar.ac` allows additional user defined toolbar items to be added in ACE. Each line is the name of a toolbar button, with the first line being the total number of buttons defined.

**ukmain.dic**

[system directory] This is the UK English dictionary, used by spell checking operations.

**user.dic**

[home directory initially, but configurable by user] This is the user's custom dictionary.

**userinfo.ac**

[system directory] This file contains a list of user name to full name mappings. The file consists of lines of the form:

*username fullname*

*username* and *fullname* are separated by whitespace.

**users.ac**

[system directory] This file contains a list of those users licensed to use the **AllChange** system. There must not be more lines (excluding comments and blank lines) than the total number of licences. Before anybody can run any **AllChange** program they must be licensed as an **AllChange** user — see [Licensing Users](#). The file consists of lines of the form:

*username* [`cr`]

Fields are separated by whitespace. If present, `cr` indicates this user is a CR-only user.

**usmain.dic**

[system directory] This is the US English dictionary, used by spell checking operations

**utility.ac**

[project then system directory] This file contains ACCEL various useful functions

**vcfunc.ac**

[project then system directory] This file contains ACCEL functions to support invoking the version control tools

**votedefs.acx**

[project directory] This file contains the definition of any votes that are defined associated with a life cycle status. This file is in XML format.

**votefunc.ac**

[project then system directory] This file contains ACCEL functions to support voting

**webdfunc.ac**

[project then system directory] This file contains ACCEL functions to support Web development/ deployment — see [Web Development Support](#).

**webmap.ac**

[project directory] This file contains mappings between local/ development and remote/ live HTML links during Web development — see [Web Development Support](#).

*local-url remote-url*

Fields are separated by whitespace.

**wizfunc.ac**

[project then system directory] This file contains ACCEL functions to support wizard dialogs — see [Wizard Dialogs](#).

**wordfunc.ac**

[project then system directory] This file contains ACCEL functions to support running **AllChange** reports from within Microsoft Word — see [Integration with Microsoft Word](#).

**workspcs.ac**

[project then system directory] This file details the workspaces known to **AllChange**. It must be present for the workspace aspects of **AllChange** to function — see [Workspaces](#). The file consists of lines of the form:

*ws-name ws-directory ws-users ws-part flags*

Fields are separated by whitespace. *Ws-users* is a comma-separated list of users. To allow for individual entries containing large numbers of usernames, entries may be continued across multiple lines by placing a \ (backslash) character as the very last character on a line; leading indentation on subsequent lines is skipped. *Flags* may be one or more of: *hierarchical*, *autoupdate*.

**wspcfunc.ac**

[project then system directory] This file contains ACCEL functions to support workspace and pool manipulation

**\*.rep**

[home then project then system directory] Files with a `.rep` suffix are textual report format files — see [Report Formats](#).

**\*.acr**

[home then project then system directory] Files with a `.acr` suffix are ACReport format files — see [Report Formats](#).

The following system files should not be tailored but are listed for information purposes:

**acacts.log**

[project directory] If this file is present the text of all actions executed by any user through the top-level **AllChange** interpreter via ACC and ACE is logged to it. **AllChange** may also be configured to log any errors to this file too. See Database.

**acreq.dot**

[Word startup directory] This file contains Microsoft Word macros to support requirements analysis facilities — see Create CR from Requirements Document.

**acuifunc.ac**

[system directory] This file contains sample functions for examining Visual ACCEL facilities — see [Visual ACCEL](#).

**allchang.dot**

[Word startup directory] This file contains Microsoft Word macros to support various aspects of Word integration — see [Integration with Microsoft Word](#).

**allchang.wiz**

[Word templates directory] This file is a Microsoft Word wizard to support running **AllChange** reports from within Word — see Running Reports from MS Word.

**blarch.exp**

[project] This file is used to support the archiving of baselines.

**cmdmap.ac**

[system directory] This file describes attributes of **AllChange** commands. It is used by ACCONFIG.

**conflist.ac**

[system directory] This file contains sundry information used by ACCONFIG.

**convfunc.ac**

[system directory] This file contains functions used in upgrading between different versions of **AllChange** — see [Upgrading to a New Version of AllChange](#).

**crarch.exp**

[project] This file is used to support the archiving of CRs.

**depends.ac**

[current directory] A file created during the build process containing information about dependencies. Used internally by the **AllChange** system.

**desired.ac**

[current directory] (Will bear a `_bt` suffix and/ or reside in a `BT` subdirectory according to operating system.) A file created during the build process containing information about the desired build thread. Used internally by the **AllChange** system.

**filelist.acu**

[system directory] This file contains information for use by ACUPGRAD — see [The Upgrade Tool – ACUPGRAD](#).

**muen.ac**

[system directory] This file contains **AllChange** licensing information. It is required for all **AllChange** functionality.

**olefunc.ac**

[system directory] This file contains sample functions for examining **AllChange** OLE Automation facilities.

**paarch.exp**

[project] This file is used to support the archiving of parts.

**pooldirs.ac**

[current directory] A file created during the build process containing information about what pool directories to use. Used internally by the **AllChange** system.

**version.ac**

[system directory] This file contains information about the current **AllChange** version. It is used in the upgrade process — see [The Upgrade Tool – ACUPGRAD](#).

**\*.exe, \*.dll**

[executable directory] These are the executables and supporting DLLs for **AllChange** programs. They must be present for these programs to work.

**\*.men**

[executable directory] These files describe the menus used by interactive programs (ACE, ACCONFIG, etc.) They must be present for these programs to work.

**\*.chm**

[executable directory] These are the help files used by interactive programs (ACE, ACCONFIG, etc.) They must be present for the help systems to work.

**\*.vcr**

[system directory] These files are used by the **Vcrep** tool.

## Appendix D - Windows Operating System Information

### Appendix D — Windows Operating System Information

- All programs bear the suffix `.EXE`.
- All programs are invoked by typing the program name in either lower or upper case.
- File patterns are those recognised by Windows, i.e. `?` and `*`.
- Long filenames and filenames with embedded spaces are supported. Filenames with spaces need quoting (with `"`) when passed to operating system commands.
- Windows programs support UNC's (Universal Naming Convention) in pathnames *insofar as they are supported by the operating system*. UNC's are pathnames which look like:

```
\\servername\sharename\path\file
```

They provide a convenient alternative to drive letters where no drive letter is connected to a disk or where the drive letter would not be the same for all users, machines or times.

- VC sub-directory name is `VC`
- There is no VC suffix. VC files *must* reside in the appropriate sub-directory.
- The last character of the newline sequence (on which workfiles are split into lines) is line-feed, ASCII 10.
- VC files may not be deleted (by `DEL`) unless they have first been made read-writable by `ATTRIB -R`.
- When writing a command action to call up a sub-program, arguments are quoted by enclosing the *whole* of the option letter part and the further information part inside the double quote character, e.g.

```
putaway "-ya multi-word reason" file
```

- Environment variables are set by:

```
SET variable=value
```

There should be no spaces on either side of the `=` symbol. Environment variables are viewed by the command `SET` (with no arguments).

- Executables, DLLs and a few GUI support files are installed in a `WIN32` subdirectory of the **All-Change** system directory. The `ACEXEDIR` environment/`.ini` file variable points to the directory.
- **AllChange** tries to read the username in the following order of precedence:
  1. from the network using a standard Windows call
  2. from the environment variable or `.ini` file entry `USER`
  3. under NT, from the environment variable or `.ini` file entry `USERNAME`
  4. it is set to `unknown`

Usernames are case insensitive, so `fred` is equivalent to `FRED`. A username may not contain any spaces, so `JOE SMITH` must not be used (though `JOE_SMITH` could be).

- Home directory is taken from the environment variable or `.ini` file entry `HOME` if it exists, otherwise it is set to `C:\`. This should be set if more than one person is to use **AllChange**.
- Temporary directory is taken from the environment variable or `.ini` file entry `TEMP` if it exists, otherwise from the environment variable or `.ini` file entry `TMP` if it exists, otherwise it defaults to `C:\TMP`.

# 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

---

	*	
*.rep		466
	.	
.rep file		207
	?	
??_status_log_arb1		302
??_status_log_class		302
??_status_log_date		302
??_status_log_item		302
??_status_log_status		302
??_status_log_user		302
	<b>A</b>	
abort		315
ac.ini		79, 206, 452
AC4AC		82, 264-265, 269, 275
acacts.log		171, 292, 466
acc		418
ACCEL		187, 220, 236, 282
ACCEL, condition		289
debugging		245
expression		283
factor		289
field references		293
functions		312
functions, dangerous		416
functions, user-defined		415
operators		289
statement		283
syntax notation		282
variables		418, 437
variables, user-defined		417
visual ACCEL		426

---

accel_profile	316
accel_warnings	316
accept changes	7
Accepted	62
Access 97	64, 164
access control	147
access field (report format)	308
Access From	84
access permissions	40
access, command access	154
lockout times	150
role definitions	152
user groups	149
user registration	147
ACCONFIG	4, 147
ACCTIMEOUT	54
ACDBGWIN	245
ACDDESERVER	79
ACDDETOPIC	79
ACDIR	79, 419
acdwscls.dll	71, 453
ACE	227, 229, 232, 237, 426
aceclipse.dll	66
acedit	6, 8
options	12
acefunc.ac	200, 225, 452
acexedir	79, 418
ACheader	450
achelpdir	79, 418
ACHOME	79
achomedir	418
ACident	450
ACInetd	30, 33
ACINETD	27
ACINTDLL.DLL	326
acjob.acx	452
acjobcs.acx	209
ACJOBS	212

acjobtimes.acx	213, 452
AClog	450
ACMAPIEX.DLL	46
ACMD	111-112, 114, 117, 122, 132
ACpart	450
acprojdir	80, 418
acproject	418
acprojtemplate	419
acprojtemplatedesc	419
acprojtemplatedir	419
ACPROJVCDIR	39, 91, 141, 143
acregversion	419
acreq.dot	466
ACSERVER	28, 32, 51
acserverprojdir	419
acserversysdir	419
acshlext.dll	68
acspwd.ac	42, 453
acspwd.new	43
acsqprojdb	419
acsqserver	419
acsysdir	419
Action	187, 202
Action Interval	212
Actions	184
activate_app	316
Actual Effort	103
ACUIDLG.DLL	426, 453
acuifunc.ac	466
ACUPGRAD	265
acversion	419, 450
acvfs	246
acweb	419
add-ins	68
Add Progression	192
Add Status	191
Add Text Box	192
Add/Remove Columns	224

---

add_status_log	316
addbaselinevote	156
addcrvote	156
addpartvote	156
admin field (user registration)	308
admin_users	419
administrative users	53
Administrator Tools	244
admusers.ac	453
Adobe Dreamweaver	71, 165
Advanced >>	198
Align	194
Align Text	195
allchang.dot	466
allchang.wiz	466
AllChange	22, 82, 245
AllChange Editor	15
AllChange, configuration facilities	147
DDE server	79
DDE topic	78
executable directory	78
licensing	18, 147
project directory	80, 452
startup file	206
system configuration	451
system directory	79, 452
Allow Admin Access	151
Allow baseline votes	423
Allow cr votes	423
Allow instances	423
Allow optimistic locking	159, 418
Allow part votes	423
Allow read permissions	170
Allow unlicensed users	81
Allow user branches	176
allow_optimistic_locking	419
Alter	144
alter_status_log	317

---

alterbaselinevote	156
altercrvote	156
alterpartvote	156
and	289
annotate	197
any	420
Any Role	155
Apache Tomcat	49
apostrophised field (nomenclature mapping)	309
app_minimise	320
app_restore	320
appendlistvar	317, 418
appendstrvar	317, 418
applets	48
apply_read_perms	158, 320
Approval Procedures	202
Approved	114, 125
araxfunc.ac	67, 164
Araxis	67, 276, 278
Araxis Merge	164
arb1--100 field (CR)	300
arb1--40 field (baseline)	298
arb1--arb40 field (part)	294
arb1 field (monitor)	299
arb1 field (status log)	302
Arbitrary Field	86, 94, 108, 112, 117, 223, 226, 293, 318-320
arbitrary_field_compulsory	318
arbitrary_field_datatype	318
arbitrary_field_display	318
arbitrary_field_edit	318
arbitrary_field_entrytype	319
arbitrary_field_name	319
arbitrary_field_new_values	319
arbitrary_field_oldname	319
arbitrary_field_values	320
archfunc.ac	162, 200, 243, 453
Archive	199, 243
enable	199

---

Archiving	162
ArchType	95
argument quoting	468
Artisan	64
Artisan Real Time Studio	164
Ask if more than one match	159
Assembly	102, 107, 110, 117, 131, 134, 137
Assesors (ITIL)	119
Assessment and Evaluation - ITIL	127
Assessors	127
Assign	189
assign CR	57
assignee field (CR)	300
asynchronous	6
Attach to Progression	195
Attach to Status	195
attach_pseudo_workspace	321
Attachment	301
Attachment directory	161
attachment_dimame	321
attributes	186
attributes field (class)	305
attributes field (cycle)	306
Authentication	52
Auto-progress	206
Auto Progress	186
autologon	56
autoupdate field (workspace)	305
available projects	84

## B

Back-out Plan	127
Background	192
backing up	145
Backup	88, 144
Baseline	107-108, 111, 131-132, 238, 298, 321-323
attachments	301
cycle	108, 111

---

baseline detail	298
baseline header	298
baseline_basename	321
baseline_items_affected	250, 257, 284, 303
baseline_template_text	321
baseline_text_filename	322
baseline_text_section	322
baseline_text_section_list	322
baseline_version	322
baselined_version	323
Baselines database	
fields	298
BCP.EXE	25
bl_arb1--15	298
bl_blinesaffected	299
bl_class	298
bl_comment	299
bl_date	298
bl_design	298
bl_detail_type	299
bl_filesaffected	299
bl_locked	298
bl_lockparts	298
bl_meta	299
bl_name	298
bl_obsolete	298
bl_partname	299
bl_status	298
bl_subblinname	299
bl_text	299
bl_text_raw	299
bl_toppart	298
bl_type	298
bl_user	298
blarch.exp	466
blexport.rep	453
blhxc1.rep	249
blia_itemaffecting	303

---

blinesaffected field (CR)	300
blinesaffecting_bline	323
blinesaffecting_file	323
blinessolved field (CR)	300
BlineTargetReleaseFields	178
Blocked Mail	206
blvt_arb1--40	304
blvt_comment	304
blvt_date	304
blvt_item	304
blvt_representing	304
blvt_status	304
blvt_user	304
blvt_vote	304
br_name	309
br_part	309
branch	169, 207, 238, 352, 388
Branch Add	207
Branch Delete	207
Branch Edit	207
Branch name	207
Branch Names	207
Branch part	207
branches.ac	207, 453
Browser	48, 147, 224, 232
browser.ac	453
bt_filename	323
bvt_user	156
Build	109, 133
build thread	323, 409
build.ini	453
build.tpl	453
Builder - ITIL	135
Business Executive _ITIL	135
button	241
bypasslogon	56

---

**C**

CAB (ITIL)	119
CAB Member - ITIL	135
calc_newversion	323
calendar	221
call	324
call_dde	324
call_dll	326
call_ole	327
call_stack	329
called_from	326
category field (report format)	308
cf_name	308
cf_unknown	308
cf_value	308
Change Advisory Board (ITIL)	119
Change Authority - ITIL	128, 136
Change Category (ITIL)	119
Change Cause (ITIL)	119
Change Management	86, 104, 111, 122
Change Manager - ITIL	127, 136
Change Process	96, 113, 135
Change Type	126
Change Type (ITIL)	119
check-outs database	297
check ACCEL syntax	6
Check Baseline Issues	244
check box	221
Check In	97, 114, 144, 186, 453
Check Issue Changes	244
Check Out	96, 113, 144, 186, 215, 453
check_arb_field	329
check_monitor	330
check_open_votes	332
check_rolemap	330
check_rolemap_user	330
check_valid_branch_name	331

---

check_view_edits	331
checking spelling	241
child_list	332
chkinoutfunc.ac	200
CI	112-113
cl_attributes	305
cl_classdefault	305
cl_classname	305
cl_classtype	305
cl_cr_num_fmt	305
cl_cr_tpl	305
cl_cyclename	305
cl_description	305
cl_template	305
class	
ProductionItem	109, 133
Class	86, 93, 122, 179, 184, 219, 226, 238, 305, 332
Assembly	107, 131
baseline	107
ControlledDoc	93
ControlledSource	93
CR	102
DCR	102
Derived	94
Document	93
Doors	94
ECR	102
HWComponent	94, 112
ITAsset	94, 112
ITCR	102
ProductionItem	108, 131
ReviewableDoc	93
SCR	102
SolidWorksDoc	93
Source	93
SWRelease	107
Websrc	94
WorkOrder	102, 117, 137

---

class field (baseline)	298
class field (CR)	300
class field (part)	294
class field (status log)	302
class_default	332
class_not_equal field (rolemap)	307
classdefault field (class)	305
classes field (fieldname)	307
classes field (rolemap)	307
classes.ac	453
classes_of_type	332
classname field (class)	305
classtype field (class)	305
Clean Up Upgrade Directory	280
Clear Field	244
Clear Syntax Error	9
clear_output	332
clear_secure_context	332
Client/Server	30-31, 89, 218, 420
Closed	123, 133
cmd_result	420
cmdfunc.ac	200, 437, 454
cmdmap.ac	466
cmpfiles	333
Column	249
command	333
Command Access	154, 207, 293, 308, 389
Command Definitions	157-158, 201, 307, 437, 451
command field (command definition)	307
command field (rolemap)	307
command name field (command definition)	307
command_first_time	420
command_items	420
command_line	420
command_name	420
commands.ac	157-158, 201, 437, 451, 454
comment	420
Comment	95

---

comment field (baseline)	298
comment field (check-out)	297
comp locked field (part)	294
comp obsolete field (part)	294
comp status field (part)	294
compare	278
Complete	114
Complexity	103
component	238
compression	89
Compulsory	221, 227
compulsory field (fieldname)	307
compulsory field (includes)	308
Compulsory Fields	225
concurrent	3
Concurrent user licensing	19, 148
condedit.ac	238, 454
Conditions	187, 226, 236, 283
config.ac	160, 455
Configuration	4, 92, 111, 132, 147
Configuration Item	112
Configuration Management	86
configuration management plan	159
Configuration Manager - ITIL	136
Configuration Options	159, 207
configuration options database, fields	308
confirm on delete	5
conflist.ac	466
Contains Text	221
Contains Words	219
control	334, 450
Control Panel	213
ControlledDoc	93, 102, 112, 115
ControlledSource	93, 96, 102, 112-113
ControlsTooltips	429
conv_path	334
conv_year4	334
convfunc.ac	466

---

Copy VC Files	275
copy_clipboard	334
copyfile	181, 334
copying a project	280
copyserverfile	335
copyvcfile	335
Cost	95
CR	102, 104, 111, 114, 122, 184, 238
DCR	102
ECR	102
ITCR	102
SCR	102
WorkOrder	102, 117, 137
CR attachments	301
CR Only	80
CR Only Users	80
cr, class	
cycle	111
CR, class	102
CR, email submission	46
remote creation	46
cr.tpl	182, 457
cr_arb1--100	301
cr_assignee	300
cr_blinesaffected	301
cr_blinessolved	301
cr_class	300
cr_crsaffected	301
cr_date	300
cr_filesaffected	301
cr_items_affected	250, 257, 284, 303
cr_num_fmt field (class)	305
cr_number	300
cr_obsolete	300
cr_only field (user registration)	308
cr_originator	300
cr_part_versionsolved	335
cr_partsaffected	301

---

cr_ref	300
cr_status	300
cr_summary	300
cr_template_text	337
cr_text	300
cr_text_filename	337
cr_text_raw	300
cr_text_section	338
cr_text_section_list	338
cr_toppart	300
cr_tpl field (class)	305
cr_versionssolved	301
crarch.exp	467
crbuffer	285
createdir	335
createserverdir	335
createvmdir	335
creating a new project	84
crexport.rep	457
cria_itemaffecting	303
crmwc1.rep	249
CRs Affected In Status	187
CRs database	300
attachments	301
fields	300
CRs For Baseline	108, 132, 186, 457
CRs, Import from email	162
crsaaffected field (CR)	300
crsaffecting	335
crsaffecting_bline	336
crsaffecting_cr	336
crsaffecting_file	336
CRsForBaselineIncludeLinks	178
crsforbline.ac	200
crssolved	336
crssolved_by_bline	337
CRTargetReleaseFields	178
crvt_arb1-40	304

---

crvt_comment	304
crvt_date	304
crvt_item	304
crvt_representing	304
crvt_status	304
crvt_user	156, 304
crvt_vote	304
crwizrep.tpl	249, 457
crxl*.xls	457
crxlstat.rep	263
crxltrnd.rep	263
csv	162, 243
csvfunc.ac	162, 200, 243
current directory	452
current record	293, 437
current_vote_users	338
cut_clipboard	338
cwd	420
cwp	420
cy_attributes	306
cy_cycle_type	306
cy_defprogression	306
cy_entry_action	306
cy_entry_cond	306
cy_exit_action	306
cy_exit_cond	306
cy_name	306
cy_newver	306
cy_progressions	306
cy_status_name	306
cy_status_type	306
cycfunc.ac	191, 457
cycle	108, 111, 132, 135
Cycle Editor	191
cycle field (rolemap)	307
Cycle View	96, 114, 165
cycle, baseline	108, 111
ControlledSource	111

---

cr	111
cycle_type field (cycle)	306
cyclename field (class)	305
cycles	6, 184
cycles database, fields	306
cycles.ac	187, 457

## D

dangerous	245, 416
dangerous functions	416
Data type	220
database	
check-outs	297
instances	297
issues	297
Database	291
database, baselines	298
CRs	300
initialising	91
monitors	299
parts	294
reporting on	256
statistics	245
datatype field (fieldname)	307
Date	221, 295, 297-298, 300, 302-303
Date Due	103
date format	248
Date picker	221
Date/ Time	221
date_compare	339
Date_Due	62
date_format_us	420
date_to_days	339
date_to_seconds	339
days_to_date	339
db_tran_flush	340
db_tran_flush_no_backout	340
dbadmin role	19, 357

---

dbsuperuser	213
dbsuperuser role	19, 357
DCR	93, 102, 105
DDE	324
dde_result_str	420
deactivate_app	340
Deadline	205
Debug	55, 245
DEBUGAC	80
DEBUGAC	80
Decision Mail	206
Decision override	205
Decision Type	205
Def Value	222
def_version	340
default value	227
default_index_value	340
defined	340
defprogression field (name)	306
del_status_log	341
deldir	341
Delete Check-out	244
delete item	8
deletebaselinevote	156
deletecrvote	156
deletepartvote	156
delfile	341
Deliverable	94, 109, 112, 133
Deliverables	133
delservdir	341
delservfile	341
delvcdir	342
delvcfile	342
depends.ac	467
deploy	215
Deployment - ITIL	136
Derived	94, 98
Derived Cycle	98

---

Derived Deliverable Patterns	178
Description	94
description field (report format)	308
design field (baseline)	298
desired.ac	467
Detach from Progression	195
Detach from Status	195
detail	298
Developer - ITIL	136
Development Tool Integrations	64, 75
Differencing	77, 80
diffsfunc.ac	200, 458
dir field (pool)	304
dir_exists	342
dir_size	342
Direct	217
direction	285
Directive	253
dimame	342
disabled field (report format)	308
discard changes	7
Discovered	103
Discrepancies	144
Display name	220
display_index_names	342
display_index_to_real_index	343
display_index_values	343
displayname field (fieldname)	307
DLL	326
dll_result_str	421
dmwvfunc.ac	71, 165, 458
do	283
do_dialog	343, 427
Document	93
Document Change Request	105
Don't wait for Editor	201
doorfunc.ac	76, 165, 458
Doors	75, 94, 105, 112, 117

---

DOORS	165
DoorsRequirement	76
DOORSWorkspace	76
Download Directory	215
Draft	115
dragdropfunc.ac	200, 458
Dreamweaver	71, 165
Dreamweaver Integration	180
Drop-down list	222
dup_ver field (part)	294

## E

each statement	283
EasyThings Web Server	48
ECAB	129
ECAB (ITIL)	119
ECAB Member - ITIL	136
echo	245, 344
echo_log_action	344
echo_log_error	344
echo_map	240, 344
echocommand	245
Eclipse	66, 164
Eclipse Integration	180
eclipsefunc.ac	66, 164
ECR	94, 102, 105
Edit Part Read Permissions	158, 245, 309, 344
Edit Status	192
Edit Text	195
Edit Text Box	195
edit, accept changes	7
delete item	8
discard changes	7
new item	8
save window	7
edit_part_perms	344
Editable	222
editable field (fieldname)	307

---

EditableMailMessages	174
EditFile	200
Editing	114
editor	5
else	288
elsif	288
email	46
Email	174
Emergency	126
EmergencyCAB (ITIL)	119
enable Archiving	199
Enable baseline votes	162
Enable CR votes	162
enable cycle viewer	165
Enable instances	162
enable MS Word 95 interface	199
Enable part votes	162
enable Web browser interface	199
Encryption	88
end_progressbar	344
endeach	283
enderror	288
endfor	287
endif	286
endwh	287
Engineering Change Request	105
Enlarge small diagrams to fit the page	197
Entry type	221
Entry/Exit Permissions	187
entry_action field (cycle)	306
entry_cond field (command definition)	307
entry_cond field (cycle)	306
entrytype field (fieldname)	307
env_ini_var	345
env_var	345
environment	22
environment variables	22, 393, 468
accessing	345

---

ACDDESERVER	79
ACDDETOPIC	79
ACDIR	79, 452
ACEXEDIR	79
ACHOME	79
ACPROJDIR	80, 452
DEBUGAC	80
HOME	79, 468
setting	396
TEMP	80, 468
TMP	80, 468
TMPDIR	78
USER	78, 468
USERNAME	468
error	345
error handling	289
error_map	240, 346
error_message	421
Escalation	209
Estimated Effort	103
evaluate	346
event field (monitor)	299
Event log	213
Event Viewer	213
Excel	68, 249, 263
Exchange	174
exit_action field (cycle)	306
exit_cond field (cycle)	306
Explorer	67
Export to Image	198
external, class	102
extract_token	346

**F**

facilities for administrators, ACE	243
Fail	133
false	421
fclose	346

---

fflush	346
Field	226
Field must have value	190
Field Name Definition	226
field name definitions	219, 236
field reference, qualified	293
field references	293
field tab	307
Field Tabs	223
field_tab	250, 284
fieldnames database, fields	307
fields	307
fields tab	226
file comparison	77
file pattern	468
File Stamping	177, 244
file, attachments	40, 300-301
function definition	199
save project	7
save window	7
file_browser_reread	347
file_exists	347
file_exists_in_projdir	347
file_exists_in_sysdir	347
file_exists_in_templatedir	347
file_locked	348
file_size	348
file_suffix	348
file_time	348
file_writable	349
filelist.acu	467
filename field (includes)	308
filename field (report format)	308
filename_part	348
filesaffected field (baseline)	298
filesaffected field (CR)	300
Fillin	221
find_executable	349

---

Firefox	49
Firewalls	55
First Project	17
Fix Field Values	244
fl_attributes	307
fl_check_spelling	307
fl_classes	307
fl_compulsory	307
fl_datatype	307
fl_displayname	307
fl_editable	307
fl_entrytype	307
fl_helpstring	307
fl_liststr	307
fl_listtext	307
fl_newname	307
fl_oldname	307
fl_span	307
fl_tabname	307
flags field (part)	294
fldnames.ac	223, 454, 458
fldtabs.ac	227, 458
floating point	291
flock	349
fm_command	307
fm_command_name	307
fm_entry_cond	307
folding	235
Follow Status Orbit	195
fopen	349
fopen_in_projdir	350
fopen_in_sysdir	350
fopen_in_templatedir	350
foreach statement	287
foredit field (check-out)	297
format_date	350
format_number	350
FormatFldName	224

---

FormatFldNameLabel	225
FormatItemTextAsHtml	63
freadline	351
frompathname	421
ft_classes	307
ft_database	307
ft_fields	307
ft_name	307
FTP	43, 215, 218
ftpfunc.ac	458
ftphost field (pool)	304
ftphost field (workspace)	305
FTPINT.DLL	44
ftpnames field (user registration)	308
ftpos field (pool)	304
ftpos field (workspace)	305
ftppath field (pool)	304
ftppath field (workspace)	305
ftppool.ac	219, 389, 458
ftpuser.ac	459
ftpwspc.ac	390, 459
full install	3, 22
Full Name	221
Full Users	80
full_to_username	351
full_to_usenamelist	351
fullname field (user registration)	308
Functions	67, 199, 308, 312
user-defined	324
fwriteline	351
fwritestr	351

**G**

Generate URL for selection	246
generate_depends	421
GenerateURL	63, 246
GenerateURLForSelectedItem	63
generic_cr_cycle	102, 105

---

Get Version to Directory	215
get_branch_list	352
get_cnum_resource_value	353
get_first_voter_for_user	353
get_next_item	353
get_part_filter	353
get_prev_item	353
get_selected	354
get_selected_issue	354
get_selected_list	354
get_selected_list_db	354
get_selected_part	355
get_system_flag	355
get_upfront_window	355
get_window_field	355
get_window_item	357
getarg	352
getarg_arb	352
getserverfile	356
getservertime	356
gettime	357
getting started	4
getver.ac	200
Global_Derived_Deliverables_Patterns	109, 133, 159
Global_SendMailFromUser	163, 174
go live	265, 279
Go Live With Other Project	280
Going Live	279
gp_location	309
gp_mailname	309
gp_name	309
gp_users	309
graphical statistics	59
Grid	197
group_list	421
Groups	149, 152, 309
groups.ac	459

---

GUI, browsers	232
condition editor	236
field names	220
menu actions	231
menu conditions	230
menu items	227

**H**

has_perm	357
has_role	358
header	298
Header Only	109
Help String	222
helpstring field (fieldname)	307
hide statuses not valid for progression	165
hierarchical field (workspace)	305
HOME	79
home directory	79, 452, 468
homedir	421
Host computer	216
Host OS	216
HP Testdirector	72, 165
HTML	48, 59, 63, 174
htmlfunc.ac	51, 161, 459
https	56
HWComponent	94, 102, 112
hyperlink	221

**I**

IBM Websphere Application Server	48
ic_comment	308
ic_compulsory	308
ic_filename	308
ic_in_use	308
Idle	96, 114
if statement	287
ifelse	358
ignoreprotecttext	157

---

ignorevote	159, 206
IIS	49
Implementation Plan	127
import, CRs	71
ini file	436
Import, CRs from email	46, 162
Import, Votes from email	46, 162
ImportFromEmail	46, 162
ImportFromEmailAllowVote	163
ImportFromEmailMailbox	46, 163
ImportFromEmailMailSystem	46, 162
ImportFromEmailNotForAllChange	47, 163
ImportFromEmailPop3Server	162
ImportFromEmailReply	47, 163
Importing CRs from CSV File	162, 243
Importing from Subversion	162
Importing Parts from CSV File	162, 243
in	283
in_date	297
in_obsolete	297
in_part	297
in_use field (includes)	308
in_user	297
Incident	117, 122
Include this Project in Job Scheduling	87, 212
includes database, fields	308
includes.ac	200, 459
Increment Revision	189
index, user-defined	249
index_names	358
ini_var	359
Initial Mail	206
install	3, 22, 30
client/ server	31
Explorer integration	67
full	3
MCSCCI	64
Word interface	68

---

workstation	3
Installation	49
instance	238
Instance	95
instance_list	359
instance_part	359
instances	162
instances database	
fields	297
Instances Database	297
instr	359
IntaChange Integration	137
Intasoft Standard	84, 92
intasoftini, import to registry	436
intaweb.ini	459
Integration	43, 252
integration with MS Word	68
Integration, Access 97	164
Integration, Araxis Merge	164
Integration, Artisan Real Time Studio	164
integration, development tools	64
explorer	67
mail	62
MCSCCI	64
MS Project	61
Powerbuilder	64
Rhapsody	75
Visual Basic	64
Visual C	64
with Excel	249, 263
with Explorer	67
with mail	62, 243
with MS Project	61
with Version Control Tools	450
with Visual Basic	64
with Visual C++	64
with Word	68, 249
Integration, Eclipse	164

---

Integration, HP Testdirector	72, 165
Integration, Microsoft Office	165
Integration, Microsoft Project	165
Integration, Rhapsody	164
Integration, SolidWorks	165
Integration, Telelogic DOORS	165
Integration, Visual Basic	164
Integration, Visual Studio	164
integrations	43
DOORS	75
Integrations, Dreamweaver	180
Integrations, Eclipse	180
Integrations, Microsoft Office	180
Integrations, Microsoft Project	181
Integrations, SCCI	181
intercept	241
Internet Explorer	49
Internet, access from	48
interpret	201, 359
interpret_success	421
Introduced	103
introduction to AllChange administration	2
Investigate	123
IR	117
is_admin_user	421
is_app_running	360
is_blocked_vote	360
is_command	360
is_cr_only_user	421
is_def_version	360
is_field_readable	158, 360, 424
is_group_user	361
is_in_baseline	361
is_in_cr_partsaffected	362
is_in_group_list	362
is_in_list	362
is_instance	362
is_issued	362

---

is_licensed_user	421
is_number	363
is_open_vote	363
is_out_for_edit	363
is_project_user	363
is_readonly	421
is_version	364
is_voter_mandatory	364
is_voter_optional	364
is_workspace_user	365
issue	238
Issue Management	86
Issue No	95, 223
issued_version	364
issues database	297
fields	297
it_asset_cycle	94, 100, 112, 116
ITAsset	94, 100, 102, 112, 116
ITCR	102, 105
item	422
item field (monitor)	299
item field (status log)	302
itemaffected	250, 257
itemaffectedtype	250, 258, 285
itemaffectingtype	303
items affected	257, 303
items_affected	250, 284
itemsaffected	303
ITIL	3, 86, 92, 111-112, 122, 131-132, 135-136
ITIL_Incident_Cycle	117, 122
ITIL_Problem_Cycle	117
ITIL_release_cycle	131
ITIL_RFC_Cycle	117
ITManagement - ITIL	136
iu_comment	297
iu_date	297
iu_foredit	297
iu_newpart	297

---

iu_optimistic	297
iu_part	297
iu_user	297
iu_wspcname	297

**J**

Java	48
Java Runtime Environment	49
Jobs	209
join	365
join_instance	365
join_parts	365
join_paths	366
join_server_paths	366
join_version	366
joinquote	366

**K**

KeepAlive	35
KeepAliveTime	35
keyword	70, 175, 292
keywords	103, 450
Keywrods	177, 244

**L**

legal_dos_pathname	367
licensed_cr_only_users	422
licensed_users	422
Licensing	3, 18, 80, 147
life-cycle	202
Life-cycle	86, 107-108, 113, 122, 184
Line Styles	195
List	221
List type	222
listlen	367
liststr field (fieldname)	307
listtext field (fieldname)	307
Live	279

---

local	367, 418
localtime_to_utc	367
Location	91, 95, 113, 141
location field (group)	309
location field (part)	294
location field (report format)	308
location raw field (part)	294
location, setting	91
variable	143
Lock Baseline	189
Lock CR	189
Locked	103
locked field (baseline)	298
locked field (part)	294
Lockout	150, 292, 389
lockout times	150
lockout.ac	292, 389, 459
lockparts field (baseline)	298
Log files	292
login	42
logon	42, 147
Logon	42, 52
Long String	221
LotusMail	63, 174
lowercase	367

**M**

Mail	46, 53, 62, 206, 243, 326
Mail (Dynamic)	188
Mail (Static)	188
Mail Import	162
mailfunc.ac	46, 162, 459
mailmap.ac	459
mailname field (group)	309
mailname field (user registration)	308
MailSystem	174
make_bt_file	368
make_user_list	368

---

MakeACInterfaceURL	63
MakeHyperlink	63
MakeWebInterfaceURL	63
Managing Files on a Remote Machine	214
Manufacturer	95, 113
map_nomenclature	240, 368
MAPI	46, 63, 162, 174
match_wild	369
max_licensed_cr_only_users	422
max_licensed_users	422
mb_abortretryignore	369
mb_btn_abort	369
mb_btn_cancel	369
mb_btn_ignore	370
mb_btn_no	370
mb_btn_notoall	370
mb_btn_ok	369
mb_btn_retry	369
mb_btn_yes	370
mb_btn_yestoall	370
mb_icon_info	369
mb_icon_question	369
mb_icon_stop	369
mb_icon_warning	369
mb_notoall	369
mb_ok	369
mb_okcancel	369
mb_yesno	369
mb_yesnocancel	369
mb_yestoall	369
MCSCCI	64, 75
Menu	227, 229, 231
actions	231
menucond.ac	231, 460
menuitem.ac	229, 460
Merge	67, 95, 276-278
message_box	369
MessageEchoMap	240

---

MessageMap	240
meta-baseline	238
meta field (baseline)	298
Microsoft Project Integration	181
Microsoft Access	64
Microsoft Access 97	64
Microsoft Common Source Code Control Interface	64
Microsoft Excel	165
Microsoft Mail	174
Microsoft Office	165
Microsoft Office Integration	180
Microsoft Project	165
Microsoft Visual Basic	64
Microsoft Visual Studio/	164
Microsoft Word	165
midlist	370
midstr	370
MinorAlpha	95
MinorNumber	95
mo_arb1	300
mo_event	300
mo_item	300
mo_user	300
Modes	191
monitor_action	201
Monitors	242, 299, 422
fields	299
Move Field	244
Mozilla	49
MS Access	64
MS Project	61
MS Word	68
mspjfunc.ac	165, 460
MSPProject Bline Start	186
MSPProject Finish	61, 186
MSPProject Start	61, 186
MSPProjectDateBaselineFinishFields	62, 165
MSPProjectDateBaselineStartFields	62, 165

---

MSPProjectNotesFromCRText	61, 165
MSPProjectResourceFromCR	61, 165
muen.ac	4, 467
Multi-select list	222
MultiLineTextToHtml	64
multisubst	371
Must Have CRs Affected	187

**N**

name field (baseline)	298
name field (configuration options)	308
name field (cycle)	306
name field (group)	309
name field (part)	294
name field (pool)	304
name field (user registration)	308
Named user licensing	18, 147
need_promote	422
need_return	422
new item	8
new project	90
New Tab	223
New Version	114
newline sequence	468
newname field (fieldname)	307
newname field (nomenclature mapping)	309
newpart field (check-out)	297
newstatus	422
newver field (cycle)	306
NFS	32
nm_apostrophised	309
nm_newname	309
nm_oldname	309
nm_plural	309
No Doc Stamp	94, 113
no_baseline_baselines_affected	422
no_baseline_files_affected	422
no_baseline_votes	423

---

no_bts	423
no_cr_baselines_affected	423
no_cr_crs_affected	423
no_cr_files_affected	423
no_cr_parts_affected	423
no_cr_votes	423
no_file field (part)	294
no_instances	423
no_monitors	423
no_part_parts_affected	423
no_part_votes	423
no_vc field (part)	294
no_ver field (part)	294
Nomenclature	238, 309, 346, 368
nomenclature mapping	157, 190, 235, 250, 256
nomenmap.ac	239, 460
Normal	125, 135
not	290
Not shown	222
not_applicable	423
Notes	95, 113
NotWorking	133
Novell	32
NoVersion	144
NT	31
num_records	371
num_records_in_range	371
num_votes	372
number field (CR)	300

## O

object	424
obsolete field (baseline)	298
obsolete field (CR)	300
obsolete field (part)	294
OffLine	95
OK to Complete CR	187
Old name	220

---

oldname field (fieldname)	307
oldname field (nomenclature mapping)	309
oldstatus	424
OLE	71, 79
OLE Automation	327
ole_result_str	424
olefunc.ac	467
OleTimeout	80, 328
onerror	288
OnLine	95
Only one version solved	115
open project	83
operating system command	333, 391
opsys	424
opsys2	424
optimistic locking	168, 181, 297, 412
option	372
option field (rolemap)	307
option_not_equal field (rolemap)	307
Options	12, 77, 190
options, check ACCEL syntax	5
configuration management plan	159
confirm on delete	5
editor	5
save settings on exit	5
toolbar	7
or	289
originator field (CR)	300
Orthogonal	194
oscommand	393
OTAServer	72
out-of-the-box configuration	92
Outlook	174
outstanding_voter_details	372
override field (rolemap)	307

**P**

pa_arb1--15	294
-------------	-----

---

pa_class	295
pa_comp_date	295
pa_comp_locked	295
pa_comp_obsolete	295
pa_comp_status	295
pa_comp_text	295
pa_comp_text_raw	295
pa_date	295
pa_dup_ver	295
pa_flags	295
pa_location	295
pa_location_raw	295
pa_locked	295
pa_name	294
pa_no_file	295
pa_no_vc	295
pa_no_ver	295
pa_obsolete	295
pa_parent	294
pa_part	294
pa_partsaffected	295
pa_pred_ver	295
pa_status	295
pa_symbname	295
pa_text	294
pa_text_raw	294
pa_type	295
pa_user	295
pa_varb1--15	294
paarch.exp	467
pachkvc	144
paexport.rep	460
pagenumber	372
paia_itemaffecting	303
parent field (part)	294
part	93-94, 111-112, 184, 207, 238
Part   Edit Branches	207
part field (check-out)	297

---

part names, legal characters	294
Part No	95
Part Permissions	309
part_browser_reread	372
part_component	373
part_filename	373
part_instance_number	373
part_items_affected	250, 257, 284, 303
part_name	373
part_name_instance	373
part_parent	374
part_pool_filename	374
part_template_text	374
part_text_filename	375
part_text_section	375
part_text_section_list	375
part_version	375
part_version_branch	376
part_version_instance	376
part_version_number	376
part_work_filename	376
part_workspace_filename	377
partclass field (baseline)	298
partname field (baseline)	298
partname field (role)	306
partname field (workspace)	305
partpermadd	158
partpermdelete	158
partpermedit	158
parts database	294
fields	294
parts, legal characters	294
relationship to files	141
parts/ files and the database	141
partsaffected	97, 114, 377
partsaffected field (CR)	300
partsaffected field (part)	294
partsaffecting	374

---

partstatus field (baseline)	298
Pass	133
password	42
paste_clipboard	377
paste_string	377
pavt_arb1--40	304
pavt_comment	304
pavt_date	304
pavt_item	304
pavt_representing	304
pavt_status	304
pavt_user	156, 304
pavt_vote	304
pawdstmp.rep	70, 460
PC-NFS	32
pc:ms-dos	214
pc:ms-unix	214
permissions	357
Permissions	187
pessimistic locking	297
PIR	129
pjNotesFromCRTText	61
plain_filename	377
plan	159
classes	179
command access	202
function definition	199
life-cycles	184
options	159
report formats	207
startup file	206
Planet Application Server	48
plural field (nomenclature mapping)	309
po_dir	304
po_ftphost	304
po_ftpos	304
po_ftppath	305
po_name	304

---

po_serverdir	304
pool	108, 131, 217, 238
pool_object	424
pooldirs.ac	467
poollist field (workspace)	305
pools	217
pools database, fields	304
pools.ac	219, 389, 460
POP3	46, 162
Populate from Predecessor	190
Populated	109, 132
port	48
port number	55, 89
pos_in_list	378
Post Implementation Review	129
Powerbuilder	64
pp_class_not_equal	310
pp_classes	310
pp_part	310
pp_recursive	310
pp_roles	310
pp_usertype	310
PR	117
Pred Baseline	108-109, 131, 190
Pred Bline	132
pred_ver field (part)	294
predecessor	108
predefined	187
Predefined Actions	188
Predefined Conditions	187
predefs.acx	191, 460
preserve case field (workspace)	305
prev_index	424
Print	197
Print Setup	197
printing, project configuration	243
Priority	103, 122
Problem	117, 122

---

Problem Management	86
Problem Source	103
process_id	424
ProcessURL	246
production_item_cycle	108-109, 131, 133
ProductionItem	102, 108-109, 117, 131, 133, 137
Progress all CRs	189
Progression Mode	196
progressions	190
progressions field (cycle)	306
Project	56, 92, 103, 111, 309, 363
project configuration	111
project configuration report	243
project directory	86
new	17
opening	83
project field (users)	308
Project Integration	181
project name	86
project selection	91
project settings editor	435
project template	92
project users	159
project, access	84
copy	90
creating new	17, 84
first	17
information	85
new	90
opening	83
updating	84
project.dic	460
project_users	378
Projected Service Outage (ITIL)	120
projects	82
projects.ac	91, 461
projfunc.ac	461
projtmpl	92

---

projtmpl.ac	92, 461
prompt	221, 378
prompt_blinelist	379
prompt_blineversionlist	379
prompt_command	380
prompt_command_values	380
prompt_condedit	380
prompt_crlist	380
prompt_dirlist	381
prompt_editfile	381
prompt_filelist	381
prompt_formatlist	382
prompt_instancelist	382
prompt_list	383
prompt_list_columns	383
prompt_partlist	384
prompt_subsyslist	385
prompt_userlist	385
prompt_versionlist	385
prompting for input	378-385
Protect Text	157
protfile	386
protserverfile	386
pseudo workspace	168, 321
PSO (ITIL)	120
putaway	144, 386, 450
putaway_cmd_result	424
putaway_store_filetime	159
putserverfile	387

**Q**

qualified field reference	293
Quality Control	104
Quantity	103
quote	387
quote_accel	387
quote_accellist	387
quote_arg	388

---

quote_arglist	388
quote_os	388
quote_oslist	388
quotelist	388
quoting	468

## R

Raised By	103
Re-read Project Configuration	244
Read-only	80, 222
Read-only Users	80
read permissions	157, 170, 320
read_branches	388
read_functions	388
read_perms_denied_message	158, 424
read_pools	389
read_report_summaries	389
read_rolemap	389
read_roles	389
read_workspaces	389
readbaseline	157, 201
readbaselinestatuslog	157, 201
readbaselinevote	156-157, 201
readcr	157, 201
readcrstatuslog	157, 201
readcrvote	156-157, 201
readinstance	157, 201
readmonitor	157, 201
readonly	421
readpart	157, 201
readpartstatuslog	157, 201
readpartvote	156-157, 201
Real Time Studio	64
Reduce large diagrams to fit the page	197
ref field (CR)	300
Refresh AC4AC Status	277
Refresh File List	277
reg_key_value_str	390

---

register definition files	242
register.ac	242, 462
registered users	152
registering users	147
registrations field (workspace)	305
registry	22, 68, 72, 78
registry editor	435
registry, ACDDSERVER	78
ACDDETOPIC	79
ACDIR	79
ACEXEDIR	79
ACHOME	79
ACPROJDIR	80
DEBUGAC	80
HOME	79
TEMP	80
TMP	80
TMPDIR	78
USER	78
regular administrative tasks	144
Rel Dir	108, 133
Rel Workspace	108, 131, 133
relative_part	390
relative_path	390
release	108, 111, 132
Release	131
release management	178
Release Management	86, 132
Release Manager - ITIL	136
Release Type	120
Release Workspace	108, 131
releasedate field (baseline)	298
ReleaseType	128
remote files	214
pools	217
workspaces	214
Remote path	216
renamecr	390

---

renameversion	391
renfile	391
renserverfile	391
renvcfile	391
rep_command_line	424
rep_str	392
repcommand	391
Report Format Syntax	283
report formats	207, 248, 252
report formats database, fields	308
report.sum	209, 462
report_format_list	392
reportfunc.ac	200, 224, 248
reports	224
Reports - ITIL	136
Request For Change	117, 122
reserving concurrent users	147
Resetting CR Counters	184
Resolution	123
Resolving Discrepancies	144
Restore	88
return statement	289
return_from_edit	425
Review	125
review-cycle	97, 115
Review Procedures	202
review_cycle	93, 178
ReviewableDoc	93-95, 97
ReviewCycles	178, 190
Reviewer	94, 113
Revision	95, 223
Revision Counter	175, 223
Revision Number	95, 189, 223
Rework	115
rf_access	308
rf_category	308
rf_description	308
rf_disabled	308

---

rf_filename	308
rf_location	308
rf_users	308
rfc	117, 122
RFC	114, 132
rfc.tpl	462
rhapfunc.ac	75, 462
Rhapsody	64, 75, 164
rinstr	392
Risk Category (ITIL)	120
rm_class_not_equal	308
rm_classes	308
rm_command	308
rm_cycle	308
rm_option	308
rm_option_not_equal	308
rm_override	308
rm_roles	308
rm_status	308
ro_partname	306
ro_role	306
ro_users	306
role	187, 389, 425
role definitions	152
role field (role)	306
role_users	392
rolemap	156, 389
rolemap database, fields	307
rolemap.ac	156, 389, 462
roles	86, 187
Roles - ITIL	135
roles database, fields	306
roles field (rolemap)	307
roles, allocating	19
dbadmin	19
dbsuperuser	19
definition	19, 152
use of	156

---

roles.ac	389, 462
RunScheduledJob	213

**S**

Save Shortcuts	243
save, project	7
window	7
saving changes	7
SCCI	164
SCCI Integration	181
sccifunc.ac	164, 463
schedfunc.ac	213
Schedule	212
Scheduled Jobs	209
SCR	93, 102, 104
scr_cycle	102
SCS	71
seconds_to_date	393
secure context	294, 416
secure_context	417, 425
Security	40
Segments	194
Select Mode	192
SendMailFromUser	174
SendVoteMailFromUser	163
Serial number	3
Serial Number	96
Serial Vote	205
server db dir	84
server name	84
Server not responding	78, 328
Server Password	147
server_env_var	393
server_oscommand	393
server_set_env_var	393
server_temp_filename	394
serverdir field (pool)	304
serverdir_exists	394

---

serverdir_size	394
serverfile_exists	394
serverfile_size	394
serverfile_time	395
serverfile_writable	395
serverpassword	42
serverprocess_id	425
Service Request	123
servlet	48
Set Font	195
Set No Keywords Substitution	244
Set System Flag	144
set_app_title	395
set_cnum_resource_value	184, 395
set_env_var	393, 396
set_file_time	396
set_ini_var	396
set_putaway_cmd_result	396
Setting up your Environment	22
setuid	48
setvar	397, 418
setvar_list	397, 418
shell_execute	397
shell_renfile	397
shell_show_file	397
shexfunc.ac	68, 463
Shortcuts	243
shortcuts.acx	463
Shortest	194
Show Field Compulsory	225
Show only project users in lists	418
show statuses made invalid by role	165
show_floating_output	398
show_floating_progress	398
show_info_list_map	240
show_obsolete	425
show_output	398
show_vars	398

---

show_window	398
showitemshiddenmessage	157
Single-select list	219
Single Sign On	52
sldworksfunc.ac	78, 165
sleep	399
SMTP	46, 54, 62-63, 162, 174
SMTPINT.DLL	46
Socket	49, 399
socket_accept	399
socket_close	399
socket_connect	399
socket_file	399
socket_server	399
SocketTimeout	55
Software Change Request	104
SolidWorks	77, 95, 101, 165
SolidWorks Show in Taskpane	223
SolidWorks Synchronise	78, 223
SolidWorksDoc	78, 93-94, 101
SolidWorksDrawing	78, 94-95, 101
sort1	425
sort2	425
sortlist	399
Source, class	93
source_cycle	93, 96, 112
Space Evenly	194
span	307
spell checking	241, 400-403
spell_engine_add_word	400
spell_engine_check_spelling	401
spell_engine_find_word	401
spell_engine_get_ignore_numbers	401
spell_engine_get_ignore_uppercase	401
spell_engine_get_language	401
spell_engine_get_suggestions	401
spell_engine_ignore_word	402
spell_engine_is_ignored_word	402

---

spell_engine_save_options	402
spell_engine_set_ignore_numbers	402
spell_engine_set_ignore_uppercase	402
spell_engine_set_language	403
splicelist	403
split	403
splitquote	403
splitunquote	404
SQL Express	23, 87
SQL Server	23
SQL Server 2008	23
SQL Server backup	146
SQL Server Browser	87
SQL Server Management Studio	23
SQL Server Permissions	28
SQL Server Roles	28
SQL_Server_Connection_Troubleshooting	27
SQLCMD.EXE	25
SQLTools	25
SQLTOOLSPATH	25
SSMS	23
Standard	92-93, 125, 420
Standard Configuration	92
standard Intasoft date format	339
startpage	56
startup file	206
statistics	245
Status	114, 184, 219, 238
Status Entry/Exit Permissions	187
status field (baseline)	298
status field (CR)	300
status field (part)	294
status field (rolemap)	307
status field (status log)	302
status log	238
Status log	212
status logs database	302
fields	302

---

Status Mode	196
status_is_open	404
status_list	404
status_name field (cycle)	306
status_type field (cycle)	306
Stop Cirteria	206
String	221
strlen	404
subblinclass field (baseline)	298
subblinname field (baseline)	298
subst	404
subsystem	238
Subversion	162, 243
SubversionCommentField	162
SubversionFullTree	162
Summary	122
summary field (CR)	300
Supplier	95, 113
svnfunc.ac	162, 244, 463
sw_release_cycle	107-108, 111
SWDoc-cycle	101
SWDoc_cycle	93-94
SWRelease	105, 107-108
symbname	425
symbname field (part)	294
syntax checking	8
sys_date	404
system configuration	451
system directory	79, 419
system files	451
ac.ini	206, 452
acacts.log	466
acjobs.acx	209
admusers.ac	453
allchang.wiz	466
browser.ac	236, 453
build.ini	453
build.tpl	453

---

classes.ac	183, 453
cmdfunc.ac	437, 454
cmdmap.ac	466
commands.ac	201, 437
condedit.ac	238, 454
config.ac	160, 455
conflist.ac	466
convfunc.ac	466
cr.tpl	182, 457
crarch.exp	467
crwizrep.tpl	249, 457
cycles	184
fldnames.ac	223, 454
ftppool.ac	219, 458
ftpuser.ac	459
ftpwspc.ac	459
groups.ac	459
includes.ac	200, 459
lockout.ac	459
mailmap.ac	459
menucond.ac	231, 460
menuitem.ac	229, 460
muen.ac	467
pools.ac	219, 460
projects.ac	91, 461
projfunc.ac	461
register.ac	242, 462
report.sum	209, 462
rfc.tpl	462
rolemap.ac	156, 462
roles.ac	462
task.tpl	463
toolbar.ac	463
version.ac	467
votedefs.acx	206
webmap.ac	219, 465
workspcs.ac	466

---

systems files	
commands.ac	454

## T

Tab	307
tabname	307
takeout	405, 450
takeout_for_edit	405
Target Release	103, 109, 120, 128, 132
task.tpl	463
Tasks	209
tdvfunc.ac	73, 165
Telelogic DOORS	165
TEMP	80
temp_dirname	406
temp_filename	394, 406
template	86, 92, 111, 181, 347, 350
Template	92
template field (class)	305
temporary Directories	54
Temporary Directories	54
temporary directory	80, 468
Web Interface	162
test	109
Test	133
Test Pool	108, 131, 133
Testdirector	165
TestDirector	72
Tester - ITIL	136
Text Box Mode	197
then	284
time out	48
timeout	80
timezone	56, 425
tmp	80
tmpdir	78, 425
Tomcat	49
toolbar	7

---

toolbar.ac	463
toolbar.ids	465
toolbar_check	425
Tooltips	427
Top Part	189
top_instance	406
topartname	425
topathname	425
TopPart	189
toppart field (baseline)	298
toppart field (CR)	300
Total Costs	121
trace	245, 407
trace output	245
trace_echo	407
trace_echo_map	240, 407
trace_map	407
transactions	
flushing	340
traperror	283
tree_depth	426
trim_spaces	407
true	426
Trusted Connection	26
Try current workspace	159
Type	103, 118, 123
type field (part)	294

**U**

ukmain.dic	465
UNC	468
unc_path	407
unformat_date	408
unformat_number	408
unix	215
UNIX - controlling files from PC	43
unknown field (configuration options)	308
Unlicensed Users	80

---

unprotfile	408
unprotserverfile	408
unquote	408
unquote_arg	408
unquote_arglist	409
unquote_os	409
unquote_oslist	409
unquotelist	409
update_bt_file	409
update_progressbar	409
update_statusbar	409
updating a project definition	84
upgrade sessions	273
upgrade stages	274
Upgrade Tool, Open Session	273
sessions	273
Stages	274
Upgrade Tool, Save Session	273
Upgrade Tool, Session Information	273
Upgrade Wizard	267
upgrading	264
Upgrading	265
uppercase	409
Urgency (ITIL)	121
URL	63
URL Link Direct to Page	56
URL Protocol	246
us_admin	309
us_cr_only	309
us_ftpnames	309
us_fullname	308
us_mailname	308
us_name	308
us_obsolete	309
us_project	309
User	18, 48, 80, 86, 147, 149, 187, 221, 308, 363, 417, 426
user field (baseline)	298
user field (check-out)	297

---

user field (monitor)	299
user field (part)	294
user field (status log)	302
User List	221
User Type	155
user.dic	465
user_to_fullname	410
user_to_fullnamelist	410
usereql	410
userinfo.ac	465
username	78
usemamelist	351
users	166
users field (group)	309
users field (report format)	308
users field (role)	306
users field (workspace)	305
users.ac	4, 453, 465
using projects	91
usmain.dic	465
utc_to_localtime	410
utilities	243
utility.ac	62, 200, 240, 246, 301, 437, 465

**V**

valid_voters	410
value field (configuration options)	308
var	411, 417
varb1–varb40 field (part)	294
variable location	141, 143
variables	417–418, 437
user-defined	324, 367, 397, 411
VC file, deleting	468
suffix	468
VC files	292
VC, sub-directory	468
vc_client_server	426
vccommand	411

---

VCControl	451
vcdir_exists	411
vcerase	411
vcfile_exists	411
vcfunc.ac	200, 437, 451, 465
vcinfo	411
VCPutaway	451
vcrep	411
vcscan	411
VCTakeout	451
vd_auto_progress	311
vd_criteria	311
vd_cycle	311
vd_deadline_field	311
vd_deadline_offset	311
vd_decision_type	311
vd_mail_blocked_body	312
vd_mail_blocked_recipient	312
vd_mail_blocked_subject	312
vd_mail_blocked_type	311
vd_mail_decision_body	311
vd_mail_decision_recipient	311
vd_mail_decision_subject	311
vd_mail_decision_type	311
vd_mail_initial_body	311
vd_mail_initial_subject	311
vd_next_word	311
vd_num_required	311
vd_serial	311
vd_status	311
vd_stop_word	311
vd_type	311
vd_voters	156, 311
version	238
Version Control Tools, integration with	450
integration with, control	334
integration with, putaway	386
integration with, takeout	405

---

integration with, vcerase	411
integration with, vcinfo	411
integration with, vcrep	411
integration with, vcscan	411
Version Control, file access controls	147
version.ac	467
version_list	412
version_successor	412
version_template_text	412
versionsolved	114
versionsolved field (CR)	300
vfsplugins.ini	67
View	186
viewer buttons	241
VisDiffs	67, 77, 80
VisMerge	67, 276, 278
Visual ACCEL	48, 343, 426
Visual Basic	64, 164
Visual C++	64
Visual Studio	64
VMS	215
VMS, controlling files from PC	43
vote	238
Vote Criteria	206
Vote Definitions	190, 202, 311
Vote Permissions	190
Vote, email submission	46
vote_deadline	413
vote_decision	413
vote_initial_mail_users	414
vote_start_date	414
voteblocked	201, 206
votedecision	201, 206
votedefs.acx	465
votefunc.ac	206, 465
voteinitiated	201, 206
voteypassnext	206
voter	238

---

Voters	204
voters_outstanding	412
Votes	156, 162, 258, 311, 332, 360, 363, 372, 410, 412-414, 423
votes database	
fields	304
Votes, Email submission	163
votes_cast	413
votes_outstanding	413
Voting	202
Voting, Cast vote by email	162

## W

Waypoints	194
Web	90, 214
web application	48
Web Browser	48, 161, 174, 215
Web Browser Access	48
client requirements	48
installation	49
registering users	53
server requirements	48
system requirements	48
Web deployment	216
Web development, support for	59
web enabled	50
web mapping	100, 219
web workspace	50, 214
web_cycle	94, 99
webdfunc.ac	59, 465
webhost field (workspace)	305
webifunc.ac	51, 161
WebInterfaceAttachmentDir	54, 161
WebInterfaceTempDir	54, 162
webmap.ac	219, 465
webos field (workspace)	305
webpath field (workspace)	305
Websrc	94, 99
What's New	1

---

while statement	287
who	243
wild_dirs	414
wild_files	414
wild_parts	414
wild_server_dirs	415
wild_server_files	415
wild_vars	415
Windows	42, 468
Windows Authentication	52
Windows Explorer, integration with	67
Windows Logon	52
WinMerge	77
winsys	426
wizard dialogs	434
wizfunc.ac	249, 434, 465
wo_cycle	102, 130, 117, 130, 137
Word	68, 249, 252
wordfunc.ac	68, 165, 466
Work Order	102, 130, 107, 110, 117, 130, 134, 137
work_filename_part	415
Working	133
WorkOrder	121
workspace	238, 452
workspace field (check-out)	297
Workspaces	20, 50, 108, 113, 214, 217, 238, 305, 389, 426, 452
web mapping	219
workspcs.ac	390, 466
workstation install	3, 22
writfile	415
wrkdir field (workspace)	305
wrkname field (workspace)	305
ws_autoupdate	305
ws_downloadaddr	306
ws_ftphost	306
ws_ftpos	306
ws_ftppath	306
ws_hierarchical	305

---

ws_partname	305
ws_poollist	305
ws_registrations	305
ws_users	305
ws_weballowinace	306
ws_webhost	306
ws_webos	306
ws_webpath	306
ws_webtransfer	306
ws_webworkspace	305
ws_wrkdir	305
ws_wrkname	305
wspcfunc.ac	200, 466
	<b>X</b>
XML	460, 465
	<b>Y</b>
Yes/ No	219
yn	415
	<b>Z</b>
Zoom	197
Zoom to Fit	197

