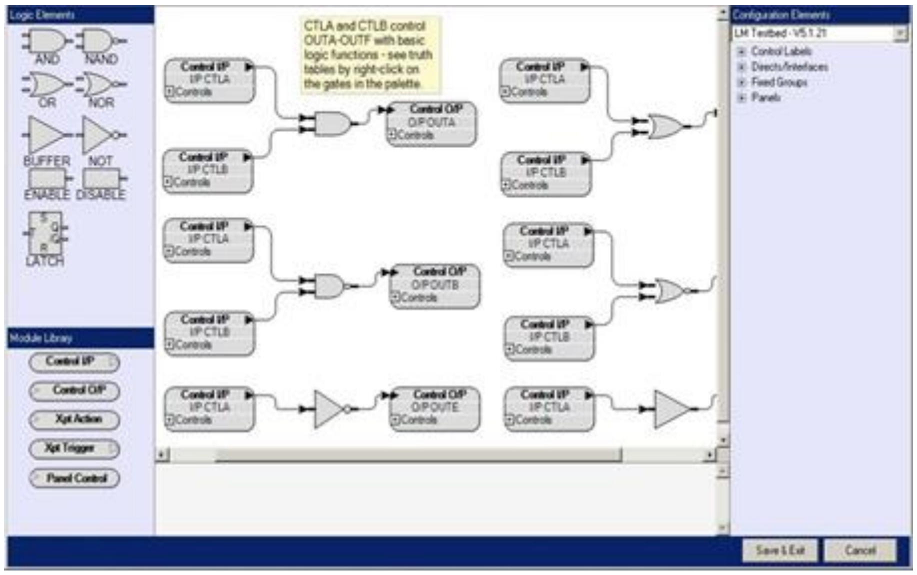
Logic


CTLA and CTLB control OUTA-OUTF with basic logic functions - see truth tables by right-click on the gates in the palette.

# Eclipse® 9.1 Logic Programming Guide

Part Number: 399G234 Rev. A

Date: 07 August, 2017

Clear-Com®
AN HME COMPANY

Document Reference

*Eclipse Logic Programming Guide*

Part Number: 399G234 Rev. A

Legal Disclaimers

Clear-Com Offices are located in California, USA; Cambridge, UK; Dubai, UAE; Montreal, Canada; and Beijing, China. Specific addresses and contact information can be found on Clear-Com's corporate website:

www.clearcom.com

Clear-Com Contacts

Americas and Asia-Pacific Headquarters
California, United States
Tel: +1 510 337 6600
Email: CustomerServicesUS@clearcom.com

Europe, Middle East, and Africa Headquarters
Cambridge, United Kingdom
Tel: +44 1223 815000
Email: CustomerServicesEMEA@clearcom.com

China Office
Beijing Representative Office
Beijing, P.R.China
Tel: +8610 65811360/65815577

## SOFTWARE LICENSE AGREEMENT

IMPORTANT–THIS IS A LEGAL AGREEMENT BETWEEN YOU AND CLEAR-COM. BEFORE DOWNLOADING, ACCESSING, OR USING ANY PART OF THE SOFTWARE, YOU SHOULD READ CAREFULLY THE FOLLOWING TERMS AND CONDITIONS CONTAINED IN THIS SOFTWARE LICENSE AGREEMENT ("AGREEMENT") AS THEY GOVERN YOUR ACCESS TO AND USE OF THE SOFTWARE. CLEAR-COM IS WILLING TO LICENSE AND ALLOW THE USE OF THIS SOFTWARE ONLY ON THE CONDITION THAT YOU ACCEPT AND AGREE TO ALL OF THE TERMS AND CONDITIONS CONTAINED IN THIS AGREEMENT. IF YOU DO NOT AGREE WITH THIS AGREEMENT, YOU ARE NOT GRANTED PERMISSION TO ACCESS OR OTHERWISE USE THE SOFTWARE.

This Agreement applies to any computer program and related files (the “Software”) offered to you by either Clear-Com LLC or HME Clear-Com Ltd. (each of whom is referred to herein as “Clear-Com”) and whether the software is delivered in the form of a diskette, DVD, USB storage device or CD-ROM (the “Storage Media”), over the Internet or through an on-line network. Your use of the Software constitutes your acceptance of the following terms and conditions.

TERMS AND CONDITIONS

1) License Grant. The Software is provided by Clear-Com, and this Agreement provides to you a personal, revocable, limited, non-exclusive, royalty-free, non-transferable license to use the Software conditioned on your continued compliance with the terms and conditions of this Agreement. This Agreement permits you to use and access for personal or business purposes the Software only at a single physical location in connection with the use of Clear-Com products. You may also load information from the Software into your laptop, workstation, or computer temporary memory (RAM) and print and download materials and information from the Software solely for your personal or business use, provided that all hard copies contain all copyright and other applicable notices contained in such materials and information. If you are using the Software on behalf of a company or other form of entity, please note that such a company or entity may have a separate agreement with Clear-Com regarding access and usage privileges for the Software. Nevertheless, your use of the Software will be subject to the obligations and restrictions regarding use of the Software as set forth in this Agreement.

2) Restrictions. The license granted under this Agreement is limited. You may not use, copy, store, reproduce, transmit, distribute, display, rent, lease, sell, modify, alter, license, sublicense, or commercially

exploit any data provided by Clear-Com through the Software in any manner not expressly permitted by this Agreement. In addition, you may not modify, translate, decompile, create any derivative work(s) of, copy, distribute, disassemble, broadcast, transmit, publish, remove or alter any proprietary notices or labels, license, sublicense, transfer, sell, mirror, frame, exploit, rent, lease, private label, grant a security interest in, or otherwise use the Software in any manner not expressly permitted herein.

3) User Obligations. By downloading, accessing, or using the Software in order to view our information and materials or submit information of any kind, you represent that you are at least 18 years of age and will, at all times, provide true, accurate, current, and complete information when submitting information or materials on the software including, without limitation, when you provide information via a software registration or submission form. In addition, you agree to abide by all applicable local, state, national, and international laws and regulations with respect to your use of the software. This Agreement is also expressly made subject to any applicable export laws, orders, restrictions, or regulations.

4) Proprietary Rights. This Agreement provides only a limited license to access and use the Software. Accordingly, you expressly acknowledge and agree that Clear-Com transfers no ownership or intellectual property interest or title in and to the Software to you or anyone else. All text, graphics, user interfaces, visual interfaces, photographs, sounds, artwork, computer code (including html code), programs, software, products, information, and documentation as well as the design, structure, selection, coordination, expression, "look and feel," and arrangement of any content contained on or available through the Software, unless otherwise indicated, are owned, controlled, and licensed by Clear-Com and its successors and assigns and are protected by law including, but not limited to, United States copyright, trade secret, patent, and trademark law, as well as other state, national, and international laws and regulations. Except as expressly provided herein, Clear-Com does not grant any express or implied right to you or any other person under any intellectual or proprietary rights. Accordingly, your unauthorized use of the Software may violate intellectual property or other proprietary rights laws as well as other laws, regulations, and statutes. The name of the Software, the Clear-Com logo, and all other names, logos, and icons identifying Clear-Com and its programs, products, and services are proprietary trademarks of Clear-Com, and any use of such marks, including, without limitation, as domain names, without the express written permission of Clear-Com is strictly prohibited. Other service and

entity names mentioned herein may be the trademarks and/or service marks of their respective owners.

5) Feedback and Submissions. Clear-Com welcomes your feedback and suggestions about Clear-Com's products or services or the Software. By transmitting any suggestions, information, material, or other content (collectively, "feedback") to Clear-Com, you represent and warrant that such feedback does not infringe or violate the intellectual property or proprietary rights of any third party (including, without limitation, patents, copyrights, or trademark rights) and that you have all rights necessary to convey to Clear-Com and enable Clear-Com to use such feedback. In addition, any feedback received will be deemed to include a royalty-free, perpetual, irrevocable, transferable, non-exclusive right and license for Clear-Com to adopt, publish, reproduce, disseminate, transmit, distribute, copy, use, create derivative works, and display (in whole or in part) worldwide, or act on such feedback without additional approval or consideration, in any form, media, or technology now known or later developed for the full term of any rights that may exist in such content, and you hereby waive any claim to the contrary.

6) Limited Warranty.

6.1 Clear-Com warrants that any Storage Media on which the Software is stored will be free from defects in materials and workmanship for 90 days from the date you acquire it. If a defect in the Storage Media occurs, return the Storage Media to Clear-Com and Clear-Com will replace it at no cost to you. This remedy is your exclusive remedy for breach of this warranty. This limited warranty is void if the damage or defect has resulted from accident, abuse or misapplication.

6.2 WHILE CLEAR-COM ENDEAVORS TO PROVIDE RELIABLE INFORMATION, SERVICES, PROGRAMS, SOFTWARE, AND MATERIALS, EXCEPT AS SPECIFICALLY PROVIDED IN SECTION 6.1, THE SOFTWARE IS PROVIDED ON AN "AS-IS" AND "AS AVAILABLE" BASIS AND MAY INCLUDE ERRORS, OMISSIONS, OR OTHER INACCURACIES. YOU ASSUME THE SOLE RISK OF MAKING USE OF THE SOFTWARE. CLEAR-COM MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE RESULTS THAT CAN BE ACHIEVED FROM OR THE SUITABILITY, COMPLETENESS, TIMELINESS, RELIABILITY, LEGALITY, OR ACCURACY OF THE SOFTWARE FOR ANY PURPOSE, AND EXPRESSLY DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OR ANY OTHER IMPLIED WARRANTY UNDER THE UNIFORM COMPUTER INFORMATION TRANSACTIONS ACT AS ENACTED BY ANY STATE. CLEAR-COM ALSO MAKES NO

REPRESENTATION OR WARRANTY THAT THE SOFTWARE WILL OPERATE ERROR FREE OR IN AN UNINTERRUPTED FASHION OR THAT ANY DOWNLOADABLE FILES OR INFORMATION WILL BE FREE OF VIRUSES OR CONTAMINATION OR DESTRUCTIVE FEATURES.

7) Limitation of Liability. You expressly absolve and release Clear-Com from any claim of harm resulting from a cause beyond Clear-Com's control, including, but not limited to, failure of electronic or mechanical equipment or communication lines, telephone or other connection problems, computer viruses, unauthorized access, theft, operator errors, severe weather, earthquakes, or natural disasters, strikes, or other labor problems, wars, or governmental restrictions. MOREOVER, IN NO EVENT SHALL CLEAR-COM BE LIABLE FOR ANY INDIRECT, PUNITIVE, INCIDENTAL, SPECIAL, EXEMPLARY, MULTIPLE, INDIRECT OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR IN ANY WAY CONNECTED WITH THE USE OF THE SOFTWARE, WITH ANY DELAY OR INABILITY TO USE THE SOFTWARE, OR FOR ANY INFORMATION, SERVICES, PROGRAMS, PRODUCTS, AND MATERIALS AVAILABLE THROUGH THE SOFTWARE, WHETHER BASED IN CONTRACT, TORT, STRICT LIABILITY, OR OTHERWISE, EVEN IF CLEAR-COM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATION MAY NOT APPLY. NOTWITHSTANDING THE FOREGOING, TOTAL LIABILITY OF CLEAR-COM FOR ANY REASON WHATSOEVER RELATED TO USE OF THE SOFTWARE SHALL NOT EXCEED THE TOTAL AMOUNT PAID BY YOU TO CLEAR-COM TO PURCHASE THE SOFTWARE.

8) Indemnity. You agree to protect, defend, indemnify, and hold harmless Clear-Com and its affiliates and all of their respective employees, agents, directors, officers, shareholders, attorneys, successors, and assigns from and against any and all claims, proceedings, damages, injuries, liabilities, losses, costs, and expenses (including reasonable attorneys' fees and litigation expenses) relating to or arising from any breach by you of this Agreement.

9) Legal Matters.

9.1 If you are resident or domiciled anywhere other than Europe, the Middle East or Africa, this Agreement will be construed and enforced solely in accordance with the laws of the State of California, U.S.A., and courts in California shall have exclusive subject matter jurisdiction, personal jurisdiction, and venue to adjudicate any dispute arising out of this Agreement.

9.2 If you are resident or domiciled in Europe, the Middle East or Africa (the “EMEA”), this Agreement will be construed and enforced solely in accordance with English law and English Courts shall have exclusive subject matter jurisdiction, personal jurisdiction and venue to adjudicate any dispute arising out of this Agreement.

9.3 You agree to waive any right to a jury trial in connection with any action or litigation in any way arising out of or related to this Agreement and acknowledge that either party may seek attorney's fees in any proceeding. Any claim you might have against Clear-Com must be brought within two (2) years after the cause of action arises, or such claim or cause of action will be barred. You also acknowledge and agree that any applicable state law implementation of the Uniform Computer Information Transactions Act (including any available remedies or laws) shall not apply to this Agreement and is hereby disclaimed.

9.4 Clear-Com makes no representation that the Software is appropriate or available for use in locations outside the State of California or the EMEA and access to or use of the Software from states, territories, or nations where any aspect of the Software is illegal is prohibited. You access or use the Software on your own volition and are responsible for compliance with all applicable local laws with respect to your access and use of the Software.

9.5 A printed version of this Agreement and of any related notice given in electronic form shall be admissible in judicial or administrative proceedings based upon or relating to this Agreement to the same extent and subject to the same conditions as other business documents and records originally generated and maintained in printed form.

9.6 Clear-Com has no obligation to provide free updates to or upgrades of the Software or to provide support of any kind with respect to the Software.

9.7 If you are acquiring the Software for an agency of the United States Government, the provisions of Federal Acquisition Regulations Section 12.212 or Department of Defense FAR Supplement Section 227.7202-3, as applicable, apply to such acquisition.

10) Term and Termination. This Agreement and your right to use the Software will take effect at the moment you click "I ACCEPT" or you install, access, or use the Software and is effective until terminated as set forth below. This Agreement will terminate automatically if you click "I REJECT." In addition, Clear-Com reserves the right at any time and on reasonable grounds, which shall include, without limitation, any reasonable belief of fraudulent or unlawful activity or

actions or omissions that violate any term or condition of this Agreement, to deny your access to the Software or to any portion thereof in order to protect its name and goodwill, its business, or other authorized users, and this Agreement will also terminate automatically if you fail to comply with this Agreement, subject to the survival rights of certain provisions identified below. Termination will be effective without notice. You may also terminate this Agreement at any time by ceasing to use the Software, but all applicable provisions of this Agreement will survive termination, as identified below. Upon termination, you must destroy all copies of any aspect of the Software in your possession. In addition to the miscellaneous section below, the provisions concerning Clear-Com's proprietary rights, feedback, indemnity, disclaimers of warranty, limitation of liability, and governing law will survive the termination of this Agreement for any reason.

11) Miscellaneous. You acknowledge that any breach, threatened or actual, of this Agreement will cause irreparable injury to Clear-Com, such injury would not be quantifiable in monetary damages, and Clear-Com would not have an adequate remedy at law. You therefore agree that Clear-Com shall be entitled, in addition to other available remedies, to seek and be awarded an injunction or other appropriate equitable relief from a court of competent jurisdiction restraining any breach, threatened or actual, of your obligations under any provision of this Agreement. Accordingly, you hereby waive any requirement that Clear-Com post any bond or other security in the event any injunctive or equitable relief is sought by or awarded to Clear-Com to enforce any provision of this Agreement. The parties agree that this Agreement is for the benefit of the parties hereto as well as Clear-Com's licensors. Accordingly, this Agreement is personal to you, and you may not assign your rights or obligations to any other person or entity without Clear-Com's prior written consent. Failure by Clear-Com to insist on strict performance of any of the terms and conditions of this Agreement will not operate as a waiver by Clear-Com of that or any subsequent default or failure of performance. If any provision (or part thereof) contained in this Agreement is determined to be void, invalid, or otherwise unenforceable by a court of competent jurisdiction or on account of a conflict with an applicable government regulation, such determination shall not affect the remaining provisions (or parts thereof) contained herein and the illegal, invalid, or unenforceable clause shall be modified in compliance with applicable law in a manner that most closely matches the intent of the original language. No joint venture, partnership, employment, or agency relationship exists between you and Clear-Com as a result of this Agreement or your utilization of the Software. Headings herein are for convenience only. This Agreement represents the entire

agreement between you and Clear-Com with respect to use of the Software, and it supersedes all prior or contemporaneous communications and proposals, whether electronic, oral, or written between you and Clear-Com with respect to the Software.

**Eclipse® 9.1 Logic Programming User Guide ........................................1**

**1 LOGIC PROGRAMMING ..................................................................13**

*1.1 INTRODUCTION.......................................................................... 13*

*1.2 OPERATION............................................................................... 13*

1.2.1 CONTROL SEQUENCE PROPERTIES ........................................... 15

1.2.2 FUNCTION BUTTONS ............................................................... 16

*1.3 LOGIC PROGRAMMING ............................................................... 20*

1.3.1 MODULE LIBRARY.................................................................... 23

1.3.2 LOGIC ELEMENTS..................................................................... 52

**2 APPENDIX A CONTROL MACRO EDITOR ..........................................65**

*2.1 INTRODUCTION TO CONTROL MACRO EDITOR ................................ 65*

*2.2 CONTROL MACRO LANGUAGE....................................................... 66*

2.2.1 EXAMPLE CONTROL MACRO ...................................................... 66

*2.3 CONTROL MACRO EDITOR ........................................................... 67*

2.3.1 CONTROL MACRO EDITOR WINDOW.......................................... 67

2.3.2 OBJECT BROWSER.................................................................... 68

2.3.3 MESSAGE WINDOW .................................................................. 68

2.3.4 RUNNING CONTROL MACROS ................................................... 68

2.3.5 STARTING THE CONTROL MACRO EDITOR.................................. 68

*2.4 CONFIGURATION ENTITIES.......................................................... 70*

*2.5 AVAILABLE MODULES.................................................................. 72*

2.5.1 CLEARCOM.............................................................................. 72

2.5.2 SHARED .................................................................................. 83

*2.6 CREATING A NEW PROJECT ........................................................ 83*

*2.7 ELEMENTS OF A CONTROL MACRO............................................... 88*

*2.8 MACRO REFERENCE ................................................................... 89*

2.8.1 ATTACHMENTOBJECT MACROS................................................. 90

*2.9 CONTROL OBJECT MACROS......................................................... 92*

2.9.1 PORT OBJECT MACROS ............................................................ 96

2.9.2 CONDITION MACROS............................................................. 100

2.9.3 CONTROL ACTIONS MACRO ................................................... 102

2.9.4 CONTROL ATTACHMENT MACROS.......................................... 107

2.9.5 CONTROL LATCH MACROS .......... 108
2.9.6 CONTROL MACROS .......... 111
2.9.7 CROSSPOINT CONTROL .......... 116
2.9.8 CURRENT MACROS .......... 117
2.9.9 LOGGING MACROS .......... 118

**3 APPENDIX B EXAMPLE CONTROL MACROS .......... 122**

*3.1 ACTIVATE SPECIFIC KEY LED .......... 122*
*3.2 ACTIVATE LED ON ALL KEYS TO DESTINATION .......... 123*
*3.3 TRIGGER ACTION WHEN BOTH A AND B ARE SET .......... 123*
*3.4 TRIGGER ACTION WHEN ALL OF A AND B AND C ARE SET .......... 124*
*3.5 CUT TALK TO STUDIO .......... 125*
*3.6 CUT TALK TO STUDIO, EXCLUDING SOME PANELS .......... 126*
*3.7 TRIGGER ACTION WHEN BOTH A IS SET AND A CROSSPOINT IS MADE127*
*3.8 TRIGGER ACTION WHEN GROUP 1 MEMBER TALKS TO GROUP 2 MEMBER 128*
*3.9 HEADSET-SELECT ON .......... 130*
*3.10 HEADSET-SELECT ON ALWAYS .......... 130*
*3.11 LOUDSPEAKER-CUT ON .......... 131*
*3.12 IFB CONTROL MACROS .......... 132*
3.12.1 ADD CALLER (IFB CHECKBOX ENABLED) .......... 132
3.12.2 ADD CALLER (IFB CHECKBOX NOT ENABLED) .......... 132
3.12.3 ADD DESTINATION .......... 132
3.12.4 ADD RETURN .......... 132
3.12.5 ADD SOURCE (IFB CHECKBOX ENABLED) .......... 133
3.12.6 ADD SOURCE (IFB CHECKBOX NOT ENABLED) .......... 133
3.12.7 EXAMPLE OF USING IFB CONTROL MACROS .......... 133

**4 APPENDIX C KEY NUMBERING ON PANELS .......... 135**

**5 GLOSSARY .......... 142**

**6 LIMITED WARRANTY .......... 147**

*6.1 WARRANTY PERIOD .......... 147*
*6.2 TECHNICAL SUPPORT .......... 147*

*6.3 WARRANTY REPAIRS AND RETURNS ............................................... 148*
*6.4 NON-WARRANTY REPAIRS AND RETURNS ....................................... 148*
*6.5 EXTENDED WARRANTY ................................................................ 148*
*6.6 LIABILITY ........................................................................... 149*

**7 TECHNICAL SUPPORT & REPAIR POLICY NOVEMBER 1, 2008 ..........150**

# 1 LOGIC PROGRAMMING

## 1.1 INTRODUCTION

The Logic facility in EHX is a separately licensable option which allows control sequences to be generated using the Logic visual programming interface. The facility to create and edit control sequence scripts directly is also available in the option via the Control Macro editor (see appendices in section 2 A and B).

Control sequences allow the configuration that controls matrix operation to be directly modified to carry out specific actions when triggered. Each control sequence contains a series of commands with each defined command representing an action carried out on an object in the configuration. An object may be a port, an input or output device or label.

The main use of control sequences is to select controls which have already been configured using EHX and modify the actions that they trigger when activated.

Each defined control sequence is named and can have multiple inputs and outputs and combination logic. These sequences take the form of actions to be associated with inputs, and the Logic editor will assist the author by providing an overview of available actions and the parameters each requires to perform the required function.

Logic elements are available (e.g. AND, NAND, OR, NOR), with tooltips supplied by the Logic editor.

It is possible that more than one control sequence in a configuration generated using Logic or the Control Macros editor may target the same action such as loudspeaker cut on a panel. This may result in one control overriding the effect of another control. For example, if two controls request loudspeaker cut on a panel, if one of the controls cancels the action it will be cancelled for both regardless of whether the other control has cancelled the action. Care should be taken to ensure that multiple controls do not target the same action to avoid unexpected results when multiple control sequences operate on the same action.

## 1.2 OPERATION

To start Logic, click on the Logic link in the Configuration menu.

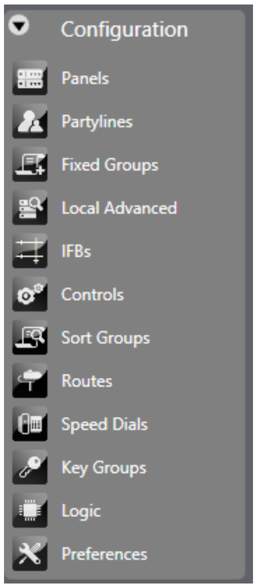
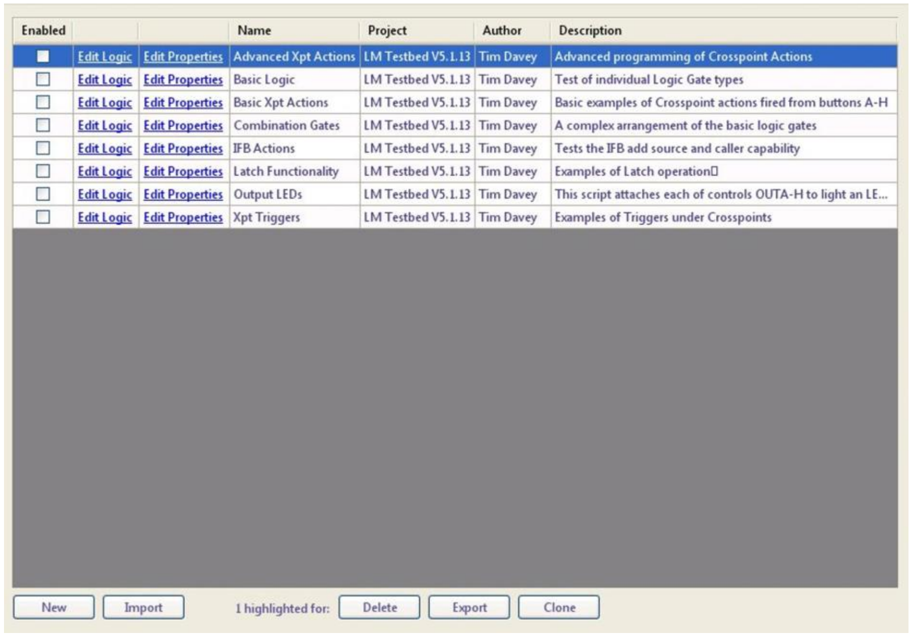


**Figure 1-1: EHX Configuration Menu**

The Logic design window will be opened displaying the initial command window with a list of known logic design. The logic design properties are displayed in seven columns.



| Enabled | | | Name | Project | Author | Description |
|---|---|---|---|---|---|---|
| ☐ | Edit Logic | Edit Properties | Advanced Xpt Actions | LM Testbed V5.1.13 | Tim Davey | Advanced programming of Crosspoint Actions |
| ☐ | Edit Logic | Edit Properties | Basic Logic | LM Testbed V5.1.13 | Tim Davey | Test of individual Logic Gate types |
| ☐ | Edit Logic | Edit Properties | Basic Xpt Actions | LM Testbed V5.1.13 | Tim Davey | Basic examples of Crosspoint actions fired from buttons A-H |
| ☐ | Edit Logic | Edit Properties | Combination Gates | LM Testbed V5.1.13 | Tim Davey | A complex arrangement of the basic logic gates |
| ☐ | Edit Logic | Edit Properties | IFB Actions | LM Testbed V5.1.13 | Tim Davey | Tests the IFB add source and caller capability |
| ☐ | Edit Logic | Edit Properties | Latch Functionality | LM Testbed V5.1.13 | Tim Davey | Examples of Latch operation□ |
| ☐ | Edit Logic | Edit Properties | Output LEDs | LM Testbed V5.1.13 | Tim Davey | This script attaches each of controls OUTA-H to light an LE... |
| ☐ | Edit Logic | Edit Properties | Xpt Triggers | LM Testbed V5.1.13 | Tim Davey | Examples of Triggers under Crosspoints |

**Figure 1-2: Logic Control Sequence List**

At the bottom of the list of control sequences there are five buttons to access functions to create, delete, import, export and clone control sequences.

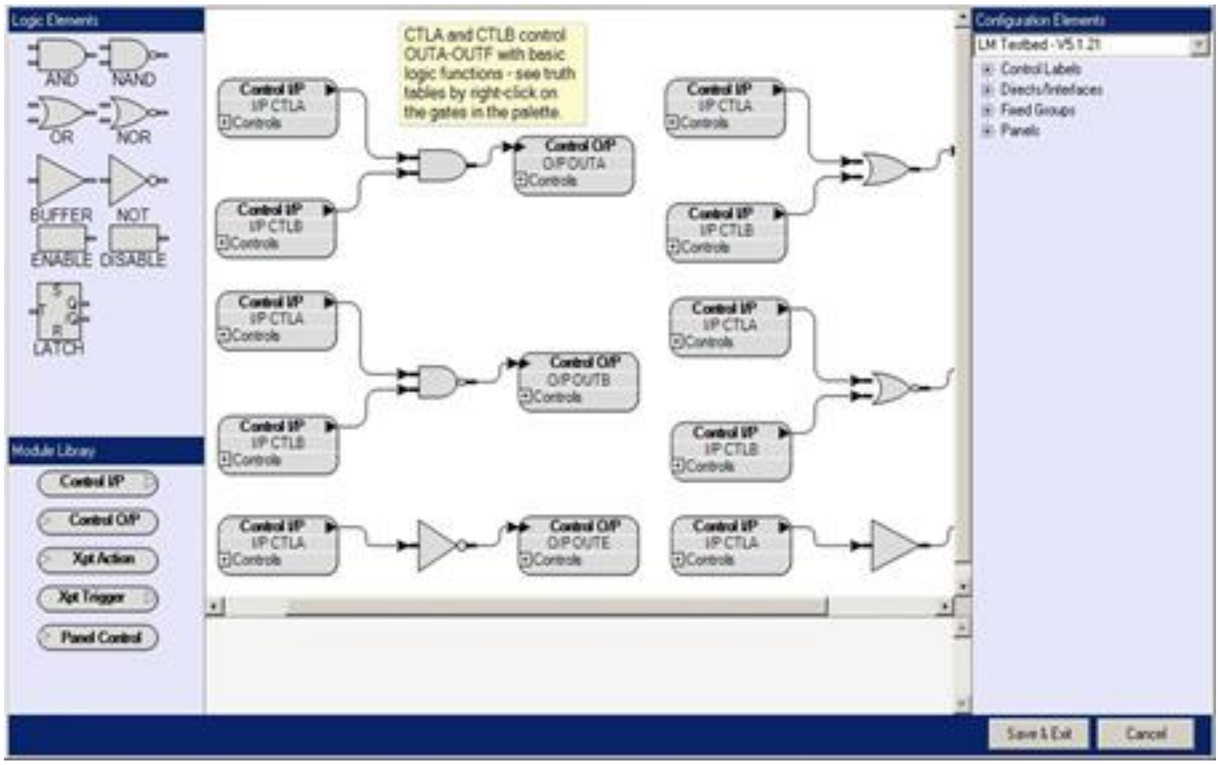
## 1.2.1 CONTROL SEQUENCE PROPERTIES

### Enabled Checkbox

The checkbox in the leftmost column of the control sequence list determines whether the Logic control sequence is saved with the system configuration in the database. If the box is checked the control sequence will be saved with the configuration; if it is not checked it will not be saved with the configuration and therefore will not be downloaded to the matrix with the configuration.

### Edit Logic Column

The 'Edit Logic' column contains links to the source for the selected control sequence. Clicking on the link will open the logic design window and display the selected control sequence in the design pane.



CTLA and CTLB control OUTA-OUTF with basic logic functions - see truth tables by right-click on the gates in the palette

**Figure 1-3: Control Sequence Display**

### Edit Properties Column

The 'Edit Properties' column contains links to the information for the selected control sequence. Clicking on this link allows the control sequence name, project name, author and description to be modified.

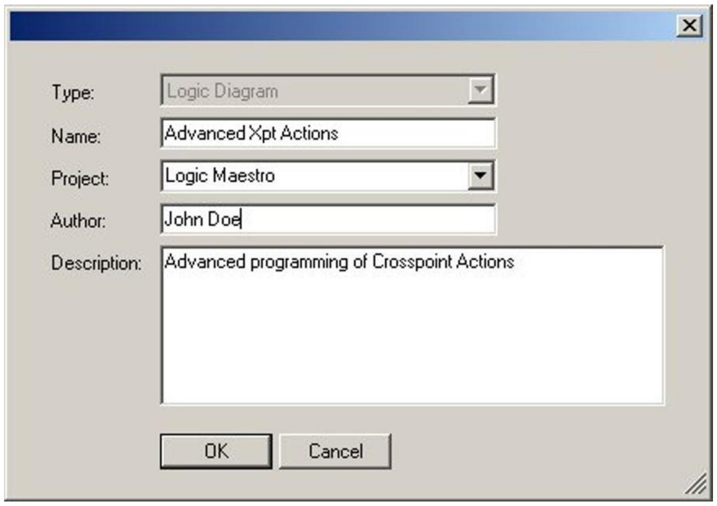

Type: Logic Diagram
Name: Advanced Xpt Actions
Project: Logic Maestro
Author: John Doe
Description: Advanced programming of Crosspoint Actions

*Figure 1-4: Control Sequence Properties*

### Name

The 'Name' column lists the names of the known control sequence designs. The control sequence design name is edited by selecting the 'Edit Properties' link for the required control sequence design.

### Project

The 'Project' column lists the project names associated with the control sequences. These project names are optional and are simply to assist in grouping control sequences together.

### Author

The 'Author' column lists the names of authors associated with the control sequences. These author names are optional and are simply for information.

### Description

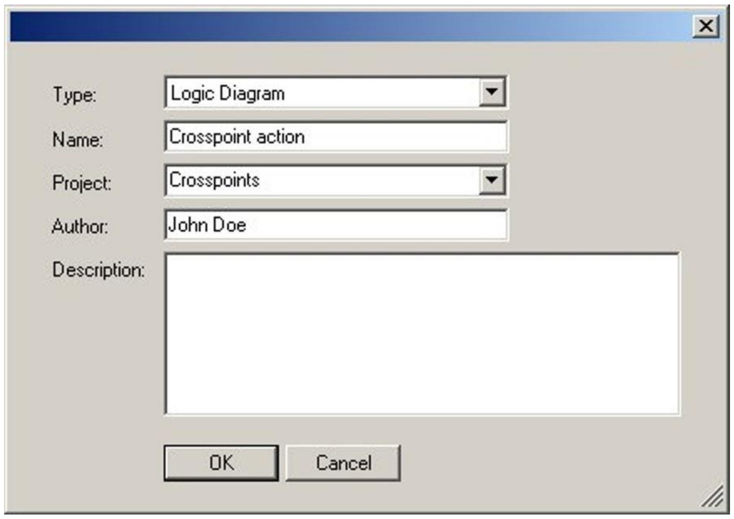
The 'Description' column lists the descriptions associated with the control sequences. These descriptions are optional and are simply for information.

## 1.2.2 FUNCTION BUTTONS

The buttons at the bottom of the logic design window allow control sequences to be created, deleted, imported from files, exported to files and cloned.

## New

Clicking the 'New' button requests the initial information for a new control sequence design, allowing the design type, design name, project name, author and a description to be input.



Type: Logic Diagram
Name: Crosspoint action
Project: Crosspoints
Author: John Doe
Description:

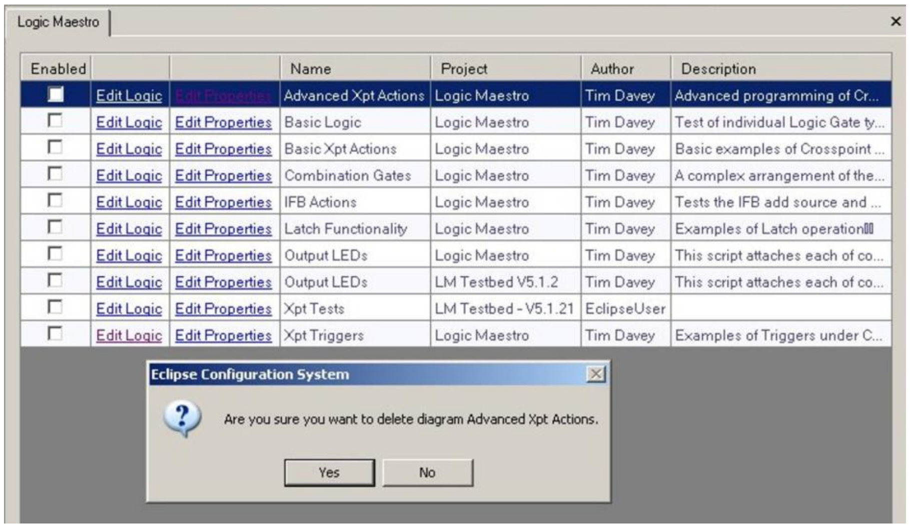
**Figure 1-5: New Control Sequence Dialog**

The 'Type' is selected from a drop-down menu and may be either 'Logic Diagram' or 'Control Macro'. Normally 'Logic Diagram' is selected and the control sequence created using the interactive design editor.

After entering the required information click on the 'OK' button to enter the Logic design environment.

## Delete

The 'Delete' button provides the facility to delete selected control sequences. Control sequences are selected for deletion by clicking on the entry to highlight it and clicking on the delete button. Multiple control sequences can be selected for deletion by pressing the 'Shift' key while selecting control sequences. A dialog is displayed to confirm the action.


Logic Maestro

| Enabled | | | Name | Project | Author | Description |
|---|---|---|---|---|---|---|
| ☐ | Edit Logic | Edit Properties | Advanced Xpt Actions | Logic Maestro | Tim Davey | Advanced programming of Cr... |
| ☐ | Edit Logic | Edit Properties | Basic Logic | Logic Maestro | Tim Davey | Test of individual Logic Gate ty... |
| ☐ | Edit Logic | Edit Properties | Basic Xpt Actions | Logic Maestro | Tim Davey | Basic examples of Crosspoint ... |
| ☐ | Edit Logic | Edit Properties | Combination Gates | Logic Maestro | Tim Davey | A complex arrangement of the... |
| ☐ | Edit Logic | Edit Properties | IFB Actions | Logic Maestro | Tim Davey | Tests the IFB add source and ... |
| ☐ | Edit Logic | Edit Properties | Latch Functionality | Logic Maestro | Tim Davey | Examples of Latch operation |
| ☐ | Edit Logic | Edit Properties | Output LEDs | Logic Maestro | Tim Davey | This script attaches each of co... |
| ☐ | Edit Logic | Edit Properties | Output LEDs | LM Testbed V5.1.2 | Tim Davey | This script attaches each of co... |
| ☐ | Edit Logic | Edit Properties | Xpt Tests | LM Testbed - V5.1.21 | EclipseUser | |
| ☐ | Edit Logic | Edit Properties | Xpt Triggers | Logic Maestro | Tim Davey | Examples of Triggers under C... |

Eclipse Configuration System

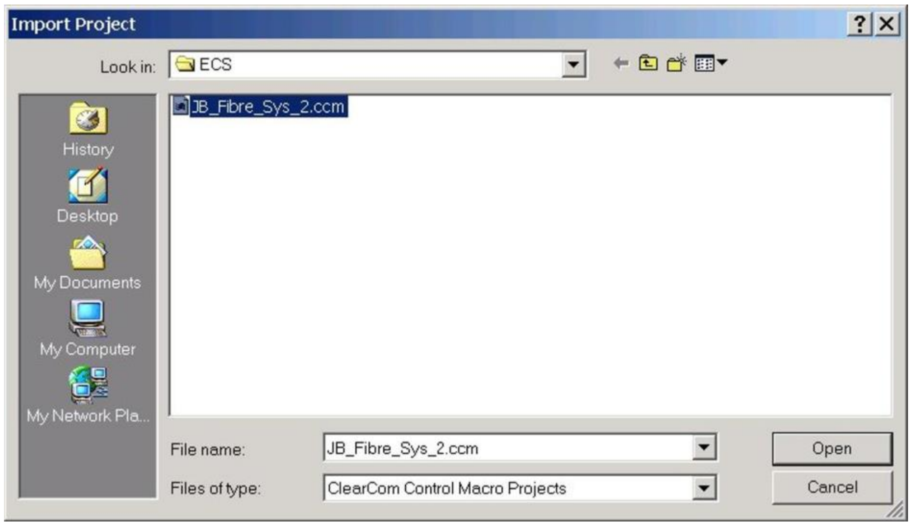
Are you sure you want to delete diagram Advanced Xpt Actions.

**Figure 1-7: Control Sequence Delete Confirmation**

Click on the ‘OK’ button to delete the control sequence.

## Import

The ‘Import’ button opens a dialogue screen to import a control sequence file (default file extension.ccm) into Logic.


Import Project

Look in: ECS

JB_Fibre_Sys_2.ccm

File name: JB_Fibre_Sys_2.ccm

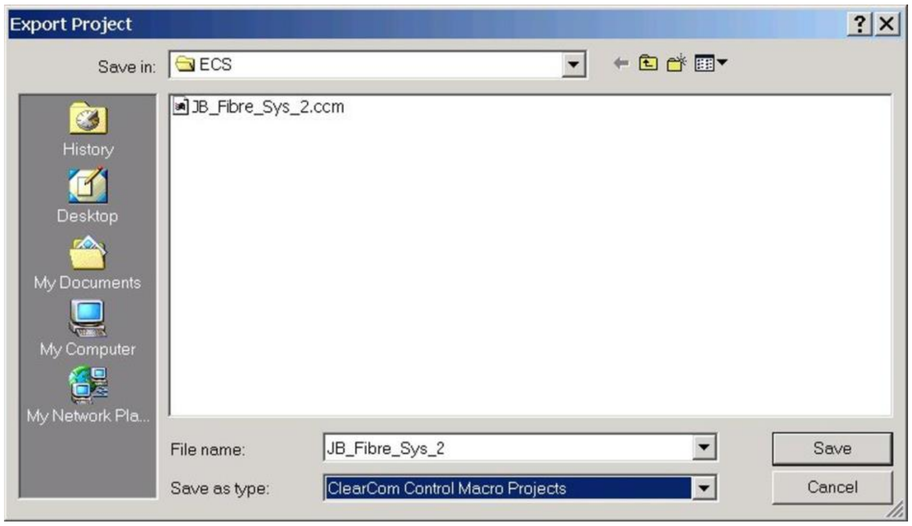
Files of type: ClearCom Control Macro Projects

**Figure 1-8: Control Sequence Import Dialogue**

Multiple control sequences can be selected for import by holding down the ’Shift’ key while selecting the control sequences to be imported.

## Export

The 'Export' button opens the dialogue screen to export a control sequence as a control sequence file. These files have a default file extension of '.ccm'. It is recommended that this default file extension is used.



File name: JB_Fibre_Sys_2

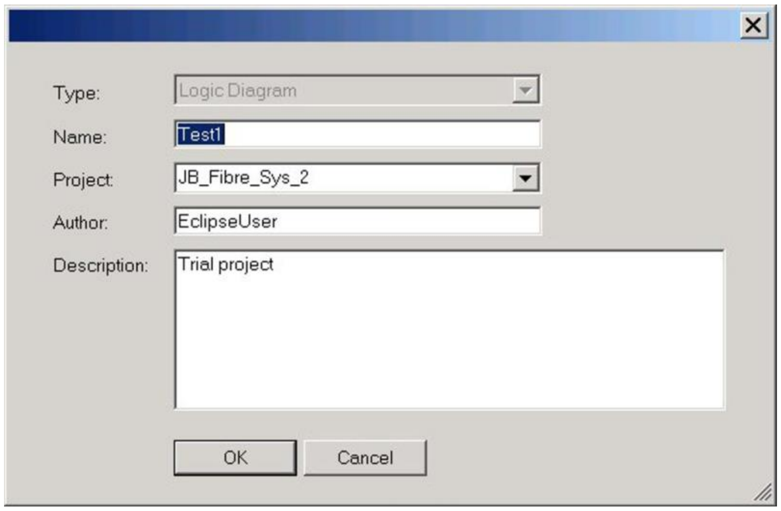
Save as type: ClearCom Control Macro Projects

*Figure 1-9: Control Sequence Export Dialogue*

Multiple control sequences can be selected for export to a single file by holding down the 'Shift' key while selecting the control sequences to be exported.

## Clone

Select the control sequence to be cloned and click on the 'Clone' button to open the control sequence clone dialog.



Type: Logic Diagram

Name: Test1

Project: JB_Fibre_Sys_2

Author: EclipseUser

Description: Trial project

**Figure 1-10: Project Clone Dialogue**

Enter a new name for the cloned control sequence, and optionally enter or change the project name, author name and description. Click on 'OK' to create the new control sequence.

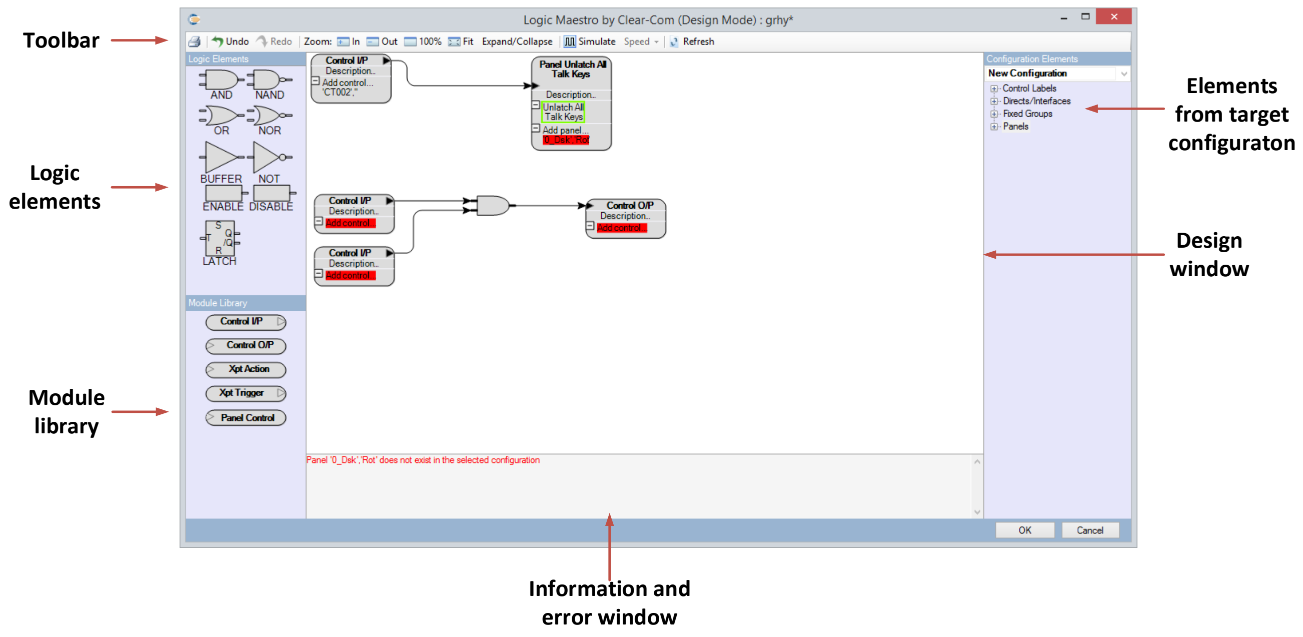
Multiple control sequences can be selected for cloning by holding down the 'Shift' key while selecting the control sequences to be cloned.

## 1.3 LOGIC PROGRAMMING

Logic allows control sequences to be created and edited by dragging and dropping logic elements and library modules onto a layout and connecting them. Configuration elements are then added to the module library elements by dragging and dropping them onto the appropriate areas of the module library elements to define the items that are to be used in the control sequence.

The toolbar allows the user to Undo and Redo changes, zoom in or out of the view, expand/collapse dropdowns, simulate inputs to the logic design, vary the speed of simulation and refresh the view.

To start a new project, click on the 'New' tab and enter the project information into the dialogue screen and click on 'OK'. The control logic layout screen is then opened.



Panel '0_Dsk','Rot' does not exist in the selected configuration

Figure 1-11: Logic Interface

Configuration elements are devices present in the target configuration (the configuration that the control sequence will be part of). These are divided into Control Inputs, Control Outputs, Directs/Interfaces, Fixed Groups, Panels and IFBs. To select a configuration element, click on the group the required configuration element belongs to and a list of all the elements in that category will be displayed in the configuration element pane.

Comments can be added to the control sequence design in two ways:

Placing the mouse pointer over the design pane but not over a design element and right clicking will create a free-floating comment box that can be edited with the text of the comment. Double click on the comment box to highlight and edit the comment text. Free floating comments can be moved around the design panel using the mouse.

Placing the mouse pointer over the title of a library module or over a logic element and right clicking will open a drop-down menu of

options. Selecting 'Add Comment' will create a comment attached to the module or logic element. Double click on the comment box to highlight and edit the comment text. Attached comments can be moved around the design panel using the mouse but will always remain connected to the target item.

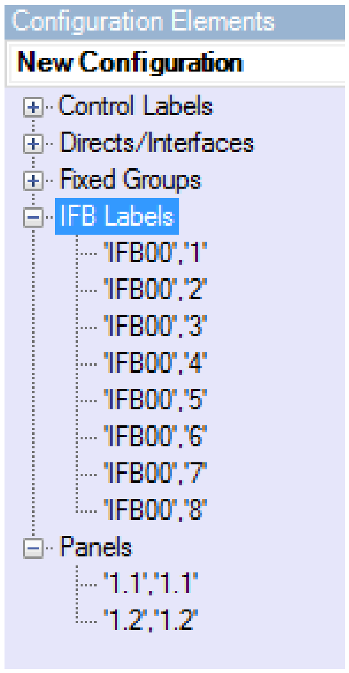


**Figure 1-12: List of Configuration Elements**

**Note:** If the configuration changes label names the Configuration elements already used in the logic diagram will show red for error. To prevent this set up fixed groups in EHX with the configuration element you need to remain static and use these for the logic diagram. The fixed group member(s) can then change names without breaking the logic.

In the case of devices with talk and listen labels both labels are displayed in the list in the format 'talk label', 'listen label'. Devices that do not have talk and listen labels are identified by name.

Logic elements can be dragged into the design pane and placed for connection to other elements.

To connect a control input to a logic element simply place the mouse pointer over the connection point on the control input, left click and hold, and drag the connection to the required connection point on the logic element and release the mouse button. The same process is used to connect the output from a logic element to a control output.

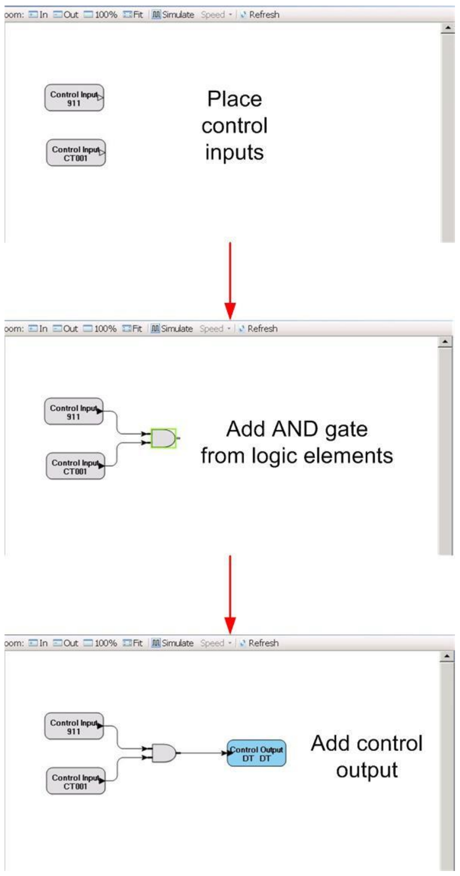


**Figure 1-13: Example Control Sequence**

Click on the 'Simulate' button on the toolbar to test the logic for errors. When simulation mode is active double clicking with the left mouse button on a logic input will invert the current state of the input unless it is an enable or disable logic element. When an element in the design is off it is colored dark grey, when on it is white. Setting an input to true allows the result of the logic design to be checked.

The speed of the simulation can be set to normal, divided by ten or divided by forty by clicking on the 'Speed' button on the toolbar and selecting the required speed from the menu. The slower speeds allow the design to be checked for race conditions that might occur if there are multiple paths between elements with different time delays in them.

Right clicking on a control sequence element will open a drop-down menu allowing the element to be deleted, cut or copied. A comment can also be added. In the case of logic elements, the type of logic element can also be changed.

## 1.3.1 MODULE LIBRARY

The module library provides control items which can be programmed with physical devices such as panel keys, direct interfaces and control labels. The physical items are then acted upon by the control items to create logic inputs and outputs, create audio paths or change the state of panel hardware.

### Control Input

Control inputs are used to provide inputs to the control sequence when a control is active. The control inputs are triggered by controls set up in EHX by the Control Manager and are usually General Purpose Inputs (GPIs). These may be attached to devices such as footswitches. Controls set up in EHX using the Control Manager may also be assigned to keys under Panel Programming. In this case activating the panel key will act as a control input.

To set up a control input drag and drop a 'Control I/P' module from the 'Module Library' onto the design pane.

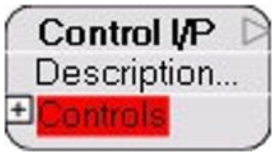


**Figure 1-14: Control Input Module**

To add a control, click on the 'Controls' menu to open it and display the 'Add control' item.

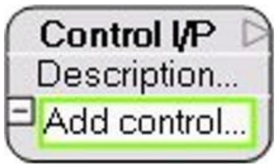


**Figure 1-15: Control Input Menu**

Drag and drop an item from the list of control labels onto the 'Add control' item to add it to the list of controls that will trigger the Control Input function.

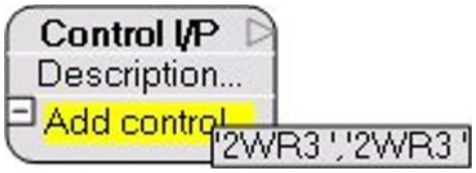


**Figure 1-16: Adding a Control to the Control Input List**

Alternatively control labels can be dragged and dropped directly onto the unexpanded 'Controls' menu and they will be added to the controls list.

Multiple control labels can be added to the control input module to create a list of control labels that will activate the logic input from the control input module.

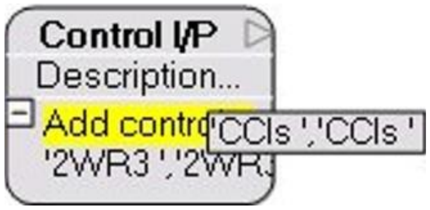


**Figure 1-17: Added Further Controls to a Control Input**

Dragging and dropping a control label on top of a label already in the controls list will replace that item with the new control label.

Items on the control list can be selected by left clicking on the items; multiple items can be selected by holding down the shift key while left clicking on the items to select them. Right clicking on the selected control item or items will open a menu giving the options to Copy, Cut Delete or Use Alt Text. Alternatively, the entire list can be copied by right-clicking on the unexpanded controls menu and selecting ''Copy this Control list'.

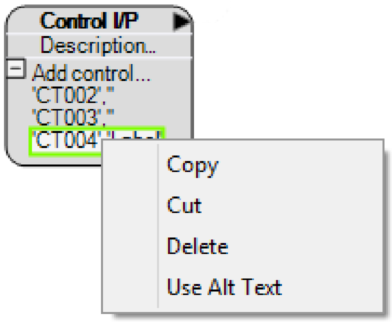


**Figure 1-18: Control List Editing**

**Note:** Note: If the control label has alternative text (Alt Text) specified in the EHX software, you can choose whether to display the label text (default) or the alternative text. Select Use Alt Text to change the text.

If items from the list of controls are cut or copied they may be pasted directly into the control list of another control input by right clicking on the unexpanded controls menu and selecting 'Add selection'.

To enter a description into the control input double left click on the word 'Description' and the description text box is displayed with the current content highlighted for overtyping.

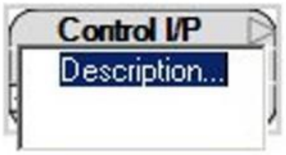


**Figure 1-19: Control Input Description**

Enter the required description in the text box and then left click outside the text box to close the text box. The description is then displayed on the control input.

A control input can be copied, cut, deleted or have a comment added by right clicking on the 'Control I/P' title to open the options menu.

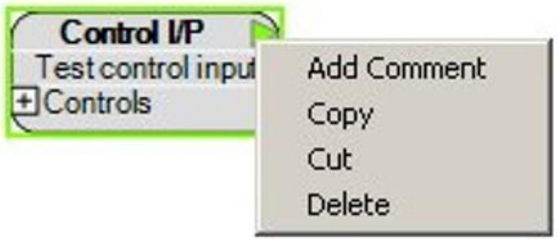


**Figure 1-20: Copying a Control Input**

If a control input is copied or cut it can be pasted back into the design window complete with the list of assigned controls and the description. Deletion will remove the control input and 'Add Comment' will add an attached comment as described previously. To paste a copy of a control input, place the mouse pointer over a free space in the design window and right click to open the menu.

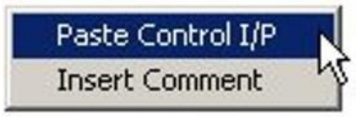


**Figure 1-21: Pasting a Control Input**

Click on 'Insert Comment' to add a free-floating comment as described previously.

## Control Input Operation

If any of the controls on the list are activated, then the control input module will be set to an active output. The same effect can be created by using multiple control inputs and combining them using 'OR' gates but whereas 'OR' gates introduce a 25ms processing delay combining multiple controls in a list does not introduce a processing delay.

## Control Output

Control outputs are used to activate outputs when the input state is true. To set up a control output drag and drop a 'Control O/P' module from the 'Module Library' onto the design pane.

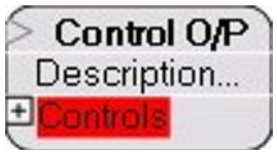


**Figure 1-22: Control Output Module**

To add a control, click on the 'Controls' menu to open it and display the 'Add control' item.

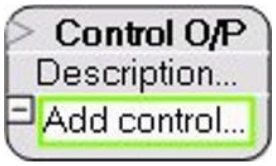


**Figure 1-23: Control Output Menu**

Drag and drop an item from the list of control labels onto the 'Add control' item to add it to the list of controls that will be triggered by the Control Output. The output control labels are set up in EHX by the Control Manager and are usually General Purpose Outputs (GPOs).

These may be attaches to external devices such as relays to control devices such as lights or door switches.

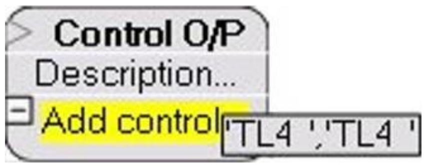


**Figure 1-24: Adding a Control to the Control Output List**

Alternatively control labels can be dragged and dropped directly onto the unexpanded 'Controls' menu and they will be added to the controls list.

Multiple control labels can be added to the control output module to create a list of control labels that will be activated by the Control Output module when it receives an active input.

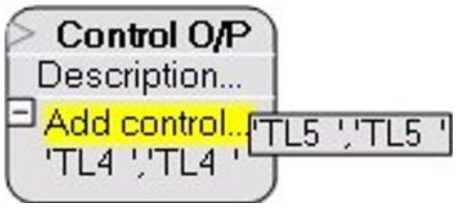


**Figure 1-25: Adding Further Controls to a Control Output**

Dragging and dropping a control label on top of a label already in the controls list will replace that item with the new control label.

Items on the control list can be selected by left clicking on the items; multiple items can be selected by holding down the shift key while left clicking on the items to select them. Right clicking on the selected control item or items will open a menu giving the options to Copy, Cut or Delete the items. Alternatively, the entire list can be copied by

right-clicking on the unexpanded controls menu and selecting ''Copy this Control list'.

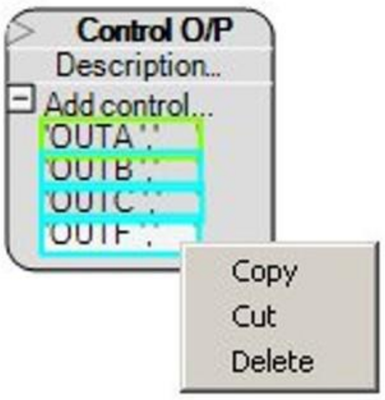


**Figure 1-26: Control List Editing**

If items from the list of controls are cut or copied they may be pasted directly into the control list of another control output by right clicking on the unexpanded controls menu and selecting 'Add selection'.

To enter a description into the control output double left click on the word 'Description' and the description text box is displayed with the current content highlighted for overtyping.

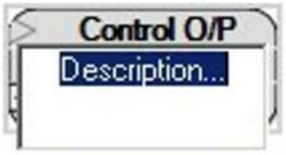


**Figure 1-27: Control Output Description**

Enter the required description in the text box and then left click outside the text box to close the text box. The description is then displayed on the control output.

A control output can be copied, cut, deleted or have a comment added by right clicking on the 'Control O/P' title to open the options menu.

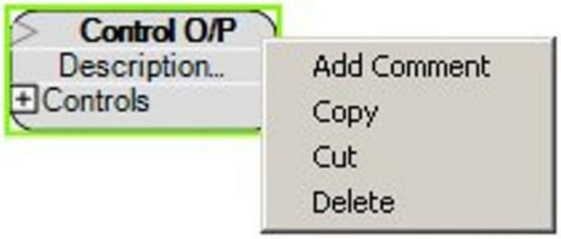


**Figure 1-28: Copying a Control Output**

If a control output is copied or cut it can be pasted back into the design window complete with the list of assigned controls and the description. Deletion will remove the control output and 'Add Comment' will add an attached comment as described previously. To paste a copy of a control output place the mouse pointer over a free space in the design window and right click to open the menu.

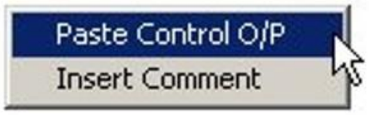


**Figure 1-29: Pasting a Control Output**

Click on 'Insert Comment' to add a free-floating comment as described previously.

Some examples of the use of input and output controls are shown in Figure 1-30 below.

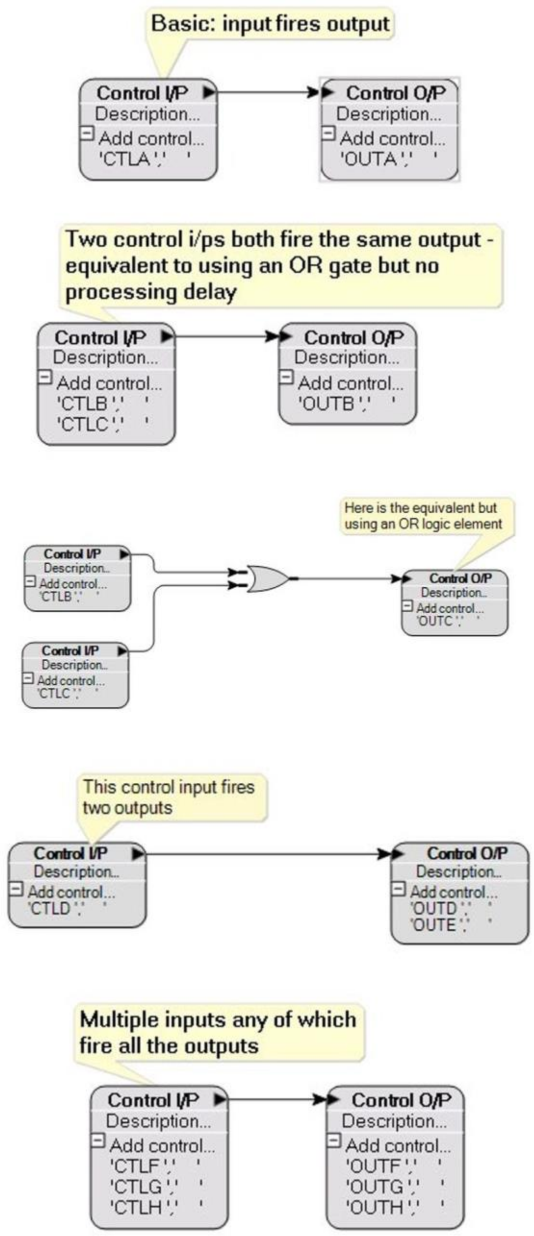


**Figure 1-30: Examples of Controls**

## Panel Control

The Panel Control module allows logic to be set up to control actions on panels and keys when the logic input is active. To set up a control output drag and drop a 'Panel Control' module from the 'Module Library' onto the design pane.

The default for a panel control is for panel loudspeaker cut.

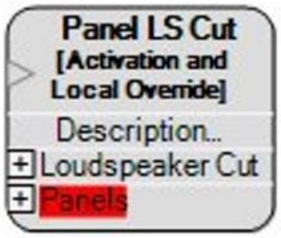


**Figure 1-31: Default Control Panel Module**

The panel control module offers the following options:

- Cut the panel loudspeaker
- Dim the panel loudspeaker
- Select the panel headset
- Mute the panel microphone
- Set the key signalization to red when active
- Set the key signalization to green when active
- Set the key signalization to amber when active
- Unlatch all talk keys
    - All V-Series panels, expansion panels and I-Series panels
    - All local and centrally assigned latched talk keys on panels and shift pages.
    - Active talk keys in interlocked groups
    - Latched keys configured as Talk, Talk & Listen and Talk & Forced Listen
    - All latched talk keys with controls or stacked assignments
    - Latched Reply keys

**Note:** The **Unlatch all talk keys** option does not apply to panel logic inputs.

To select a different option open the action menu (‘Loudspeaker Cut’) and right click on the current option to display the options list.

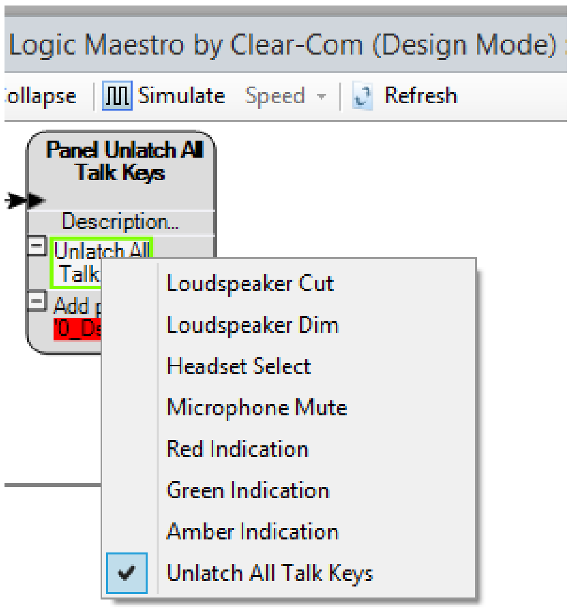


**Figure 1-32: Panel Control Options**

Select the panel control option required from the list by  left clicking on it. The list will be closed and the panel control module display will be updated according to the option selected.

If a key signalization is selected red, green, amber) the key indication on the label can be set to one of the options:

- Indication Off
- Indication 1Hz
- Indication 2Hz
- Indication 4Hz
- Indication On

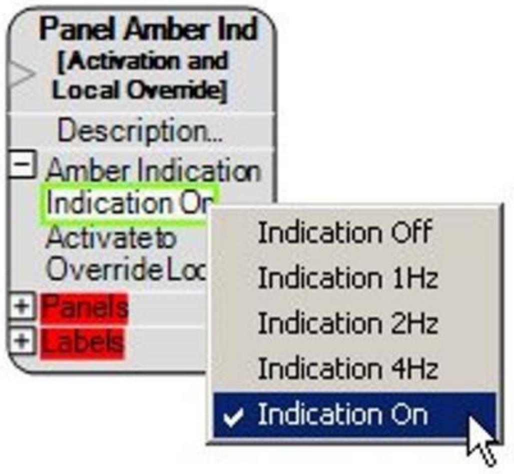


**Figure 1-33: Key Signalization Options**

The panel override options for key signalization are:

- Activate to Override Local
- Permanent Override of Local
- Advanced Override of Local

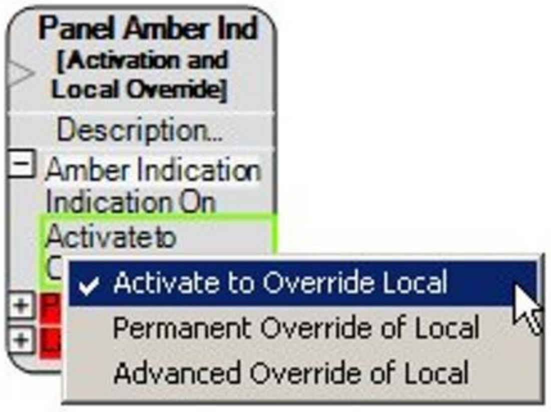

Panel Amber Ind
[Activation and Local Override]
Description...
Amber Indication
Indication On
Activate to
✓ Activate to Override Local
Permanent Override of Local
Advanced Override of Local

**Figure 1-34: Panel Override Options for Key Signalization**

Drag and drop one or more panels onto the 'Add Panel' menu to configure the panels that will be the subject of the controls. If key signalizations are required drag and drop the required control labels, Directs/Interfaces, Fixed Groups or Panels onto the 'Add Label' menu.

If a panel loudspeaker, headset or microphone action is selected the 'Labels' menu is not available. It is only available when a key signalization panel action is selected.

If loudspeaker cut, loudspeaker dim, select panel headset or panel microphone mute are set as the action the options menu for these actions are:

- Activate to Override Local. Overrides the current setting of the device if it is currently active. If it is not active the control has no effect.

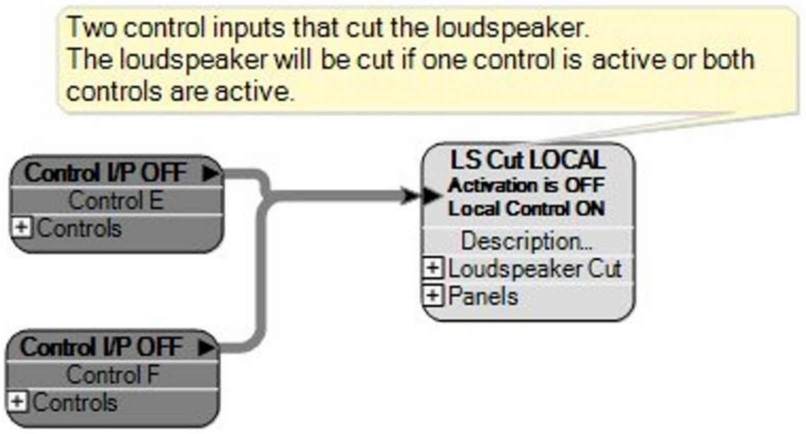


**Figure 1-35: Panel Override IF Active Example**

- Permanent Override of Local. Always overrides the current setting of the device regardless of whether it is active or not.

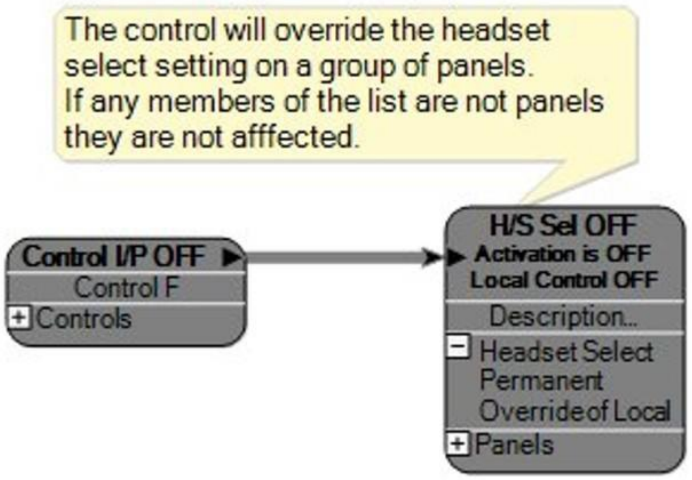

The control will override the headset select setting on a group of panels.
If any members of the list are not panels they are not afffected.
Control I/P OFF
Control F
Controls
H/S Sel OFF
Activation is OFF
Local Control OFF
Description...
Headset Select
Permanent
Override of Local
Panels

**Figure 1-36: Permanent Override of Local Example**

Advanced Override of Local. In this case there are two control inputs to the panel. The first control input must be active for the second control input to take over the panel function.

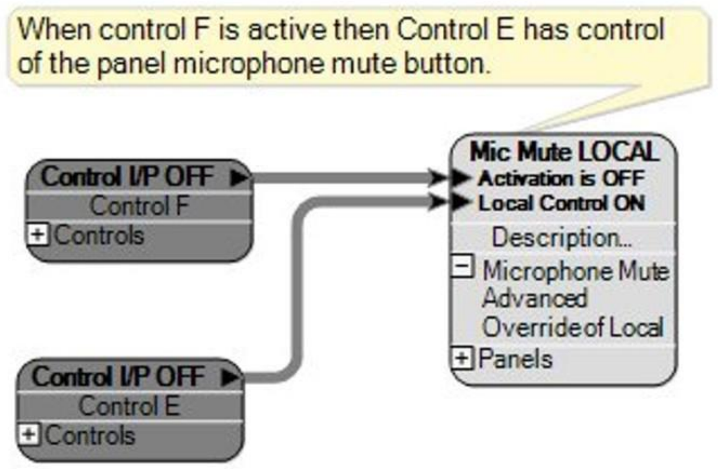


**Figure 1-37: Advanced Override of Local Example**

## Crosspoint Trigger

Crosspoint triggers allow audio crosspoints to be used to generate a control output to another action which may be a control output or a crosspoint action. Crosspoint triggers are configured with sources and destinations selected from the lists of fixed groups and panels that define the crosspoints.

To set a crosspoint trigger drag and drop an ‘Xpt Trigger’ from the ‘Module Library’ pane onto the design pane.

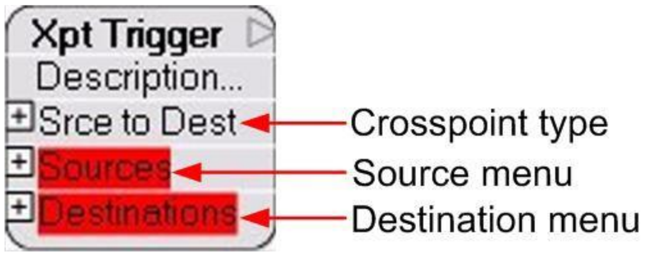


**Figure 1-38: Crosspoint Trigger**

## Trigger Crosspoint Type

Open the crosspoint type menu and right click on the current type to display the menu of trigger types.

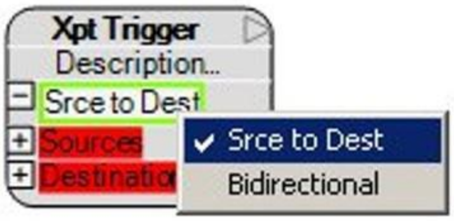


**Figure 1-39: Crosspoint Trigger Type Menu**

The crosspoint trigger can be set to operate when either source to destination crosspoints are made or bidirectional crosspoints are made between any of the sources and destinations configured. Right click on the menu item to select the crosspoint trigger type.

## Crosspoint Trigger Sources

Crosspoint trigger sources can be added to the list by dragging and dropping devices from the Direct/Interfaces, Fixed Groups and Panels lists onto the source list whether or not it is open. If the source list is opened then dropping a new source onto an existing source will replace it. If there are no items already assigned to the source list then the list name will be highlighted in red. If there are items already assigned the list will not be highlighted but instead will be surrounded by a green box.

**Note:** With Eclipse 8.7, you cannot add sources to a virtual IFB by creating a crosspoint from a source to a destination. To modify the contents of a virtual IFB, you must use a control that fires a route.

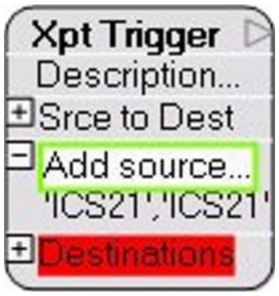


**Figure 1-40: Menu Selected**

When the menu name is highlighted in yellow the item can be dropped onto the menu.

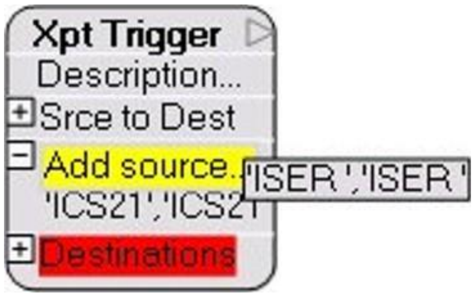


**Figure 1-41: New Item Added**

Right-clicking on 'Add source...' will display a menu allowing all the ports or all the panels in the configuration to be added to the source list.

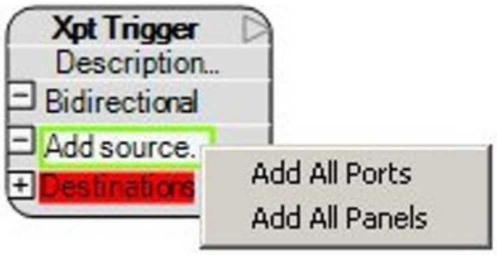


**Figure 1-42: Adding All Ports or Panels to Crosspoint Trigger**

Sources in the list can be copied, cut, deleted or excluded by selecting the required items from the list and right clicking to display the options list. Multiple items on the list can be selected by holding down the Shift key while selecting items.

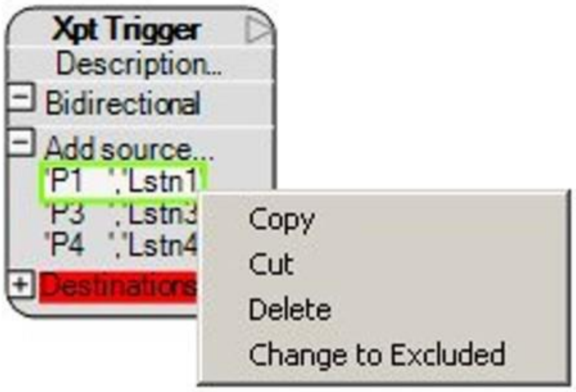


**Figure 1-43: Crosspoint Trigger Source Options**

Items that are cut or copied can be pasted into other source or destination lists. Deleting an item removes it from the list while the 'Change to Excluded' option allows a source to be excluded from consideration when triggering an output. If a source is excluded it will be displayed in red. If 'All Ports' is present in the source list this cannot be excluded.

## Crosspoint Trigger Destinations

Crosspoint trigger destinations can be added to the list by dragging and dropping devices from the Direct/Interfaces, Fixed Groups and Panels lists onto the destination list whether or not it is open. If the destination list is opened then dropping a new destination onto an existing destination will replace it. If there are no items already assigned to the destination list then the list name will be highlighted in red. If there are items already assigned the list will not be highlighted but instead will be surrounded by a green box.

**Note:** With Eclipse 8.7, you cannot set a virtual IFB as the destination of a crosspoint action. You must either use a route, or add the IFB to a fixed group, and add that group as the destination of the crosspoint action.

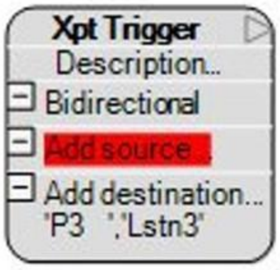


**Figure 1-44: Destination Menu Selected**

When the list name is highlighted in yellow the item can be dropped into the list.

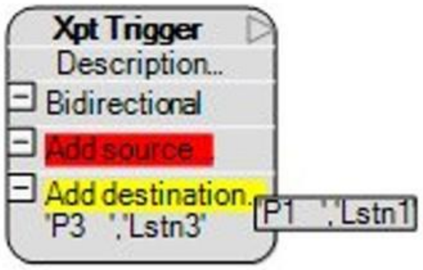


**Figure 1-45: New Item Added**

Right-clicking on 'Add destination...' will display a menu allowing all the ports or all the panels in the configuration to be added to the destination list.

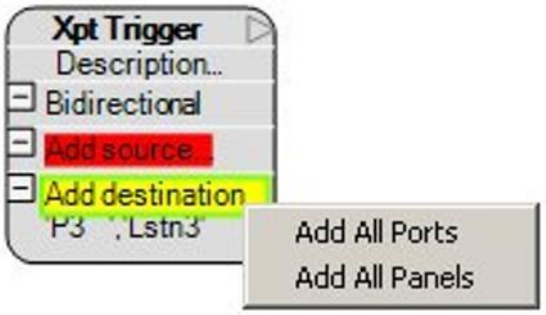


**Figure 1-46: Adding All Ports or Panels to Crosspoint Trigger**

Destinations in the list can be copied, cut, deleted or excluded by selecting the required items from the list and right clicking to display the options list. Multiple items on the list can be selected by holding down the Shift key while selecting items.

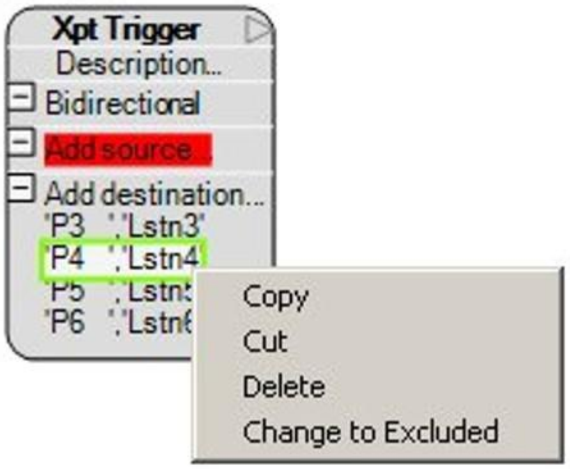


**Figure 1-47: Crosspoint Trigger Source Options**

Items that are cut or copied can be pasted into other source or destination lists. Deleting an item removes it from the list while the 'Change to Excluded' option allows a destination to be excluded from consideration when triggering an output. Any destination that has been excluded is shown in red. If 'All Ports' is present in the destination list this cannot be excluded. To re-include a destination that has been excluded select it and right click to open the actions menu and select 'Change to Included'.

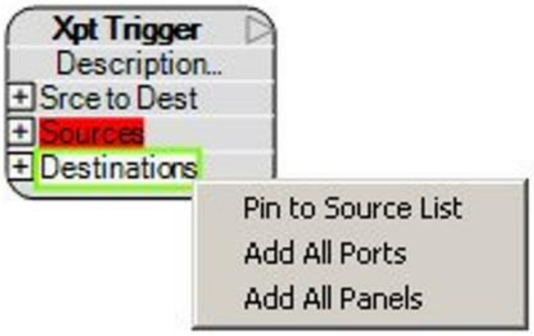


**Figure 1-48: Pin to Source List Destination Option**

Selecting the 'Pin to Source List' option replaces the 'Destinations' menu with 'Dests => Sources'.

To reinstate the 'Destinations' menu right click on 'Dests => Sources' and select 'Detach from Source List'.

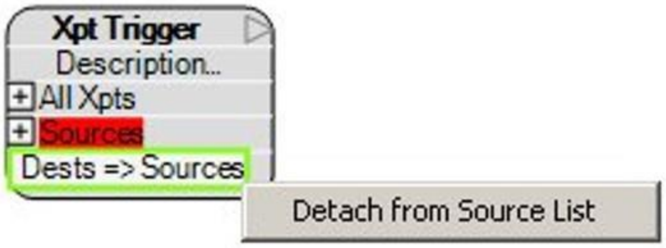


**Figure 1-49: Delete Pin to Source List Option**

The 'Dests => Sources' option replaces the destination list with a matrix of crosspoints between all the sources in the source list. This is shown by the crosspoint options menu being replaced by a new 'All Xpts' menu. Right-clicking on the 'All Xpts' menu will display a list of options allowing the crosspoint matrix to be modified.

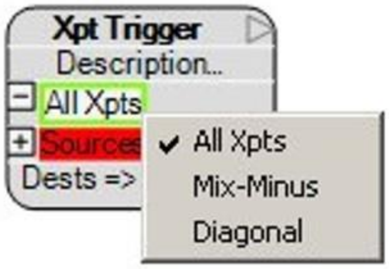


**Figure 1-50: Cross Points Options**

The crosspoint options for Pin to Source are:

- All Xpts - triggers on every crosspoint between sources in the source list. The example below shows the table for sources 1 - 6.

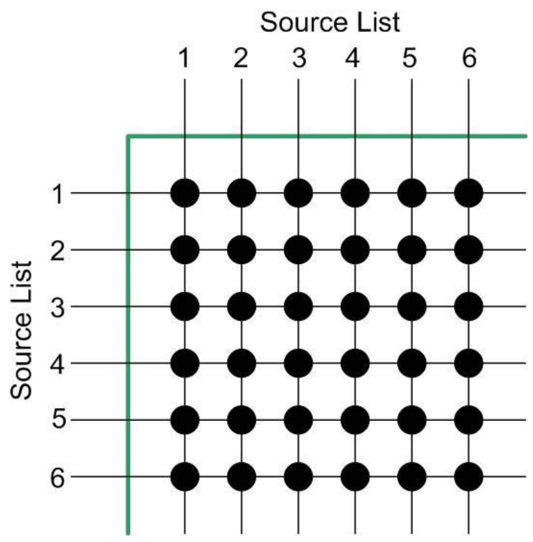


**Figure 1-51: All Possible Crosspoints Set as Trigger**

- Mix-Minus - triggers on every crosspoint between sources on the source list except loopback crosspoints that form the diagonal on the crosspoint matrix. The example below shows the table for sources 1 - 6.

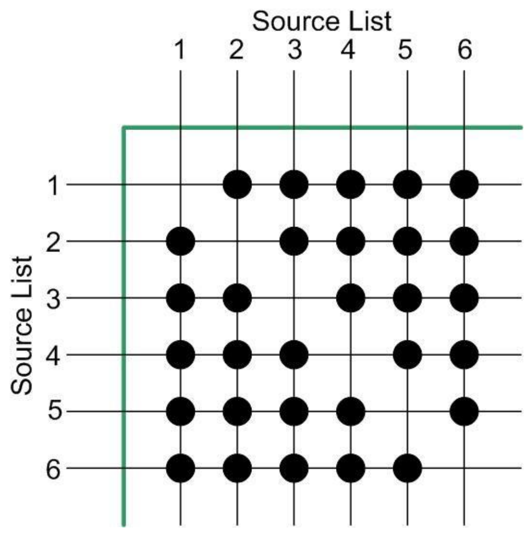


**Figure 1-52: Mix-Minus Crosspoints**

- Diagonal - triggers on all loopback crosspoints .i.e. where sources on the source list are looped back to themselves. The example below shows the table for sources 1 - 6.

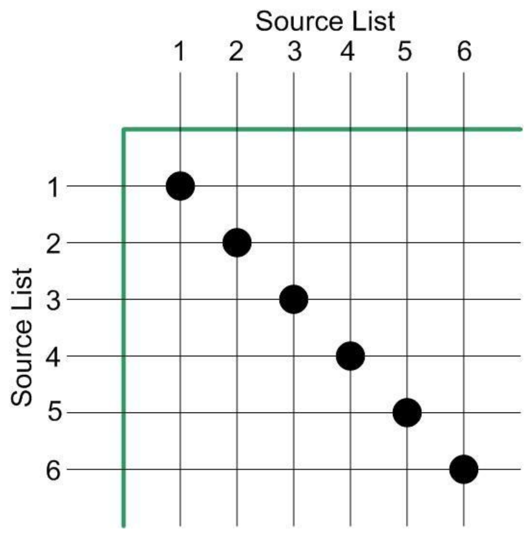


**Figure 1-53: Diagonal Crosspoints**

## Crosspoint Trigger Examples

Examples of the use of crosspoint triggers and actions are shown below.

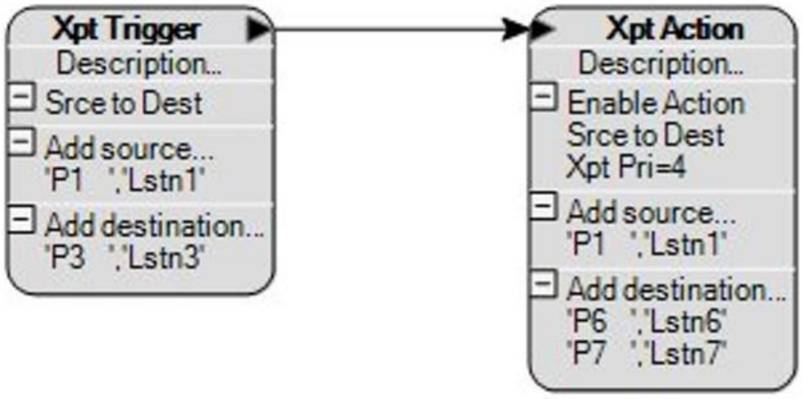


**Figure 1-54: Crosspoint Trigger for Crosspoint Action**

In example Figure 1-54 when source 'P1' establishes an audio path to destination 'P3' the crosspoint trigger will be activated to provide an input to the crosspoint action. The crosspoint action will enable crosspoints between the same source 'P1' and two other destinations 'P6' and 'P7' at priority 4.

The effect would be that whenever the panel operator 'P1' talks to 'P3' the audio will also be routed to 'P6' and 'P7'.

Examples of crosspoint triggers used to trigger control outputs are shown in Figure 1-55.

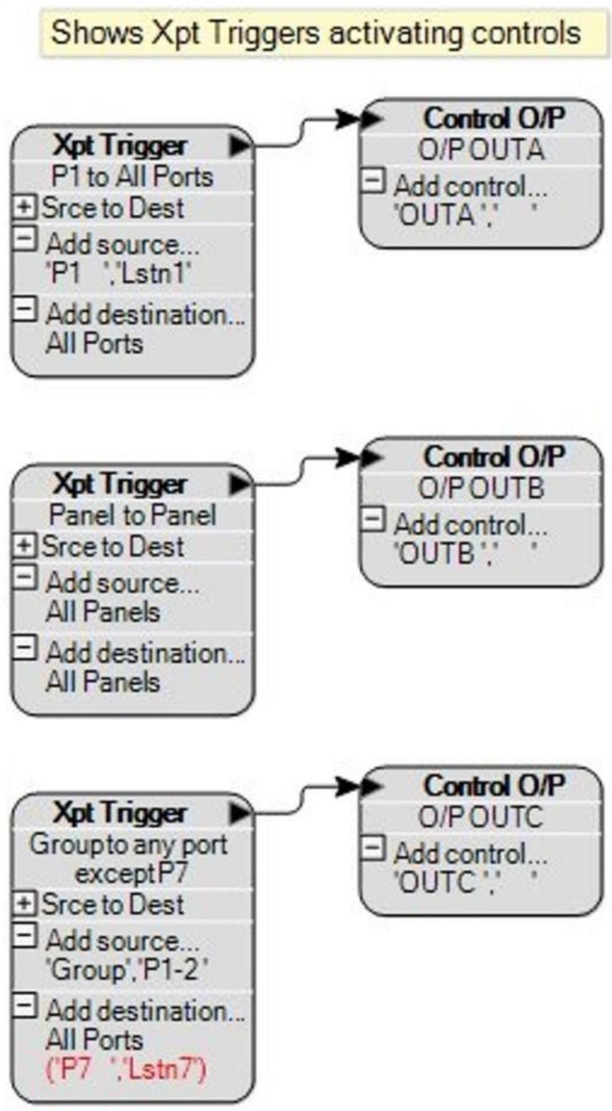


**Figure 1-55: Crosspoints Triggering Control Outputs**

Crosspoints triggers can be used to enable other crosspoints so that a single key could enable audio feeds from a number of sources to a number of destinations as shown in Figure 1-56.

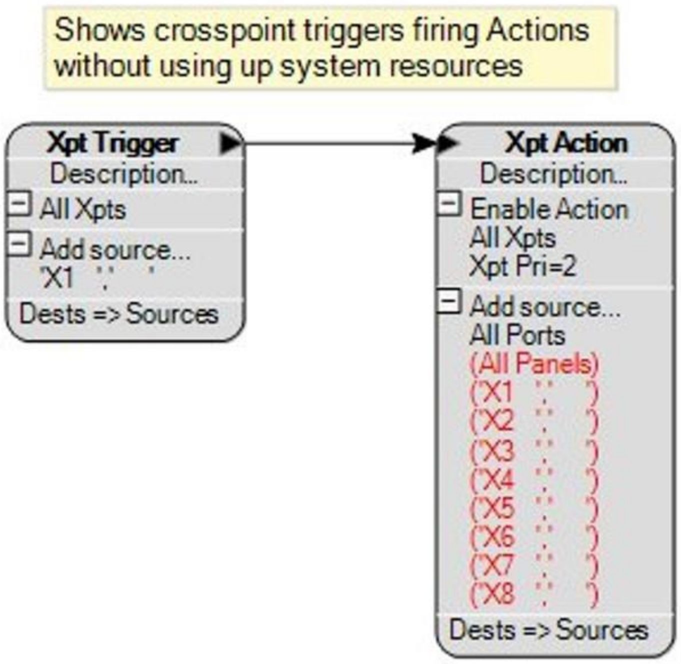


**Figure 1-56: Crosspoint Triggering Many Actions**

There is a constraint on the maximum number of possible actions by crosspoint triggers and crosspoint actions imposed by system resources. In general the number of possible triggers times the number of possible actions should not exceed 4095. So if there are 16 possible triggers specified in an Xpt Trigger and 16 possible crosspoint actions specified in a Xpt Action the number of actions would be:

16 triggers x 16 actions = 256 events

which would be acceptable. If the result of setting up a system of crosspoint triggers and crosspoint actions created more than 4095 possible actions an error would be reported when the configuration was downloaded.

In this case a buffer logic element should be placed between the crosspoint trigger and crosspoint action. In this way the number of actions the trigger crosspoint has to make is limited to the number of trigger crosspoints, which only have to trigger the buffer. The buffer will then act on the crosspoints in the crosspoint action. An example of this is shown in Figure 1-57.

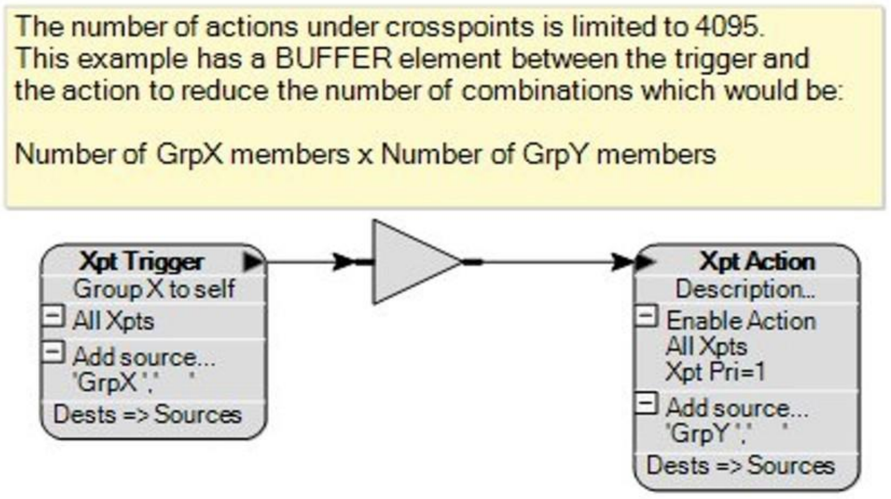

The number of actions under crosspoints is limited to 4095.
This example has a BUFFER element between the trigger and the action to reduce the number of combinations which would be:
Number of GrpX members x Number of GrpY members
Xpt Trigger
Group X to self
All Xpts
Add source...
'GrpX',' '
Dests => Sources
Xpt Action
Description...
Enable Action
All Xpts
Xpt Pri=1
Add source...
'GrpY',' '
Dests => Sources

**Figure 1-57: Many to Many Action with Buffer**

Crosspoint actions can also be triggered from control inputs either directly or through other logic elements.

## Crosspoint Action

Crosspoint actions allow crosspoint triggers or control inputs to act on audio crosspoints in various ways depending on how the crosspoint action is set up. Crosspoint actions are configured with sources and destinations selected from the lists of fixed groups and panels that define the crosspoints.

To set a crosspoint action drag and drop an ‘Xpt Action’ from the ‘Module Library’ pane onto the design pane.

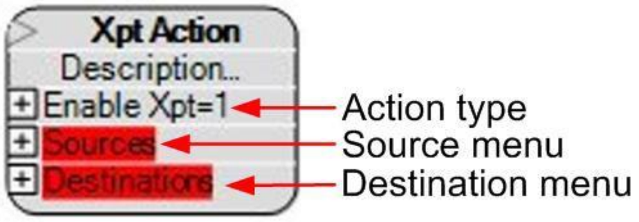


**Figure 1-58: Crosspoint Action**

## Action Type

The action type menu allows the type of action (enable, inhibit, isolate) to be specified, together with the two crosspoints to be acted on (source to destination, bidirectional) and the crosspoint priority. There is also the IFB action, specifying the source port, the destination IFB and the type of action (call, destination, source, return).

Open the action type menu and right click on the current action to display the menu of crosspoint actions.

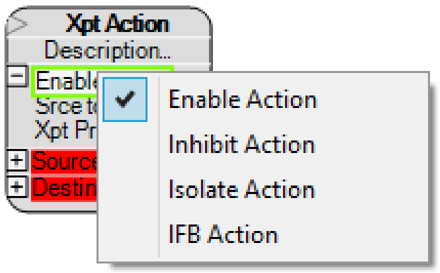


**Figure 1-59: Crosspoint Actions List**

The available crosspoint actions are:

- Enable Action - enable all the crosspoints between the sources and destinations that satisfy the crosspoint type and priority criteria except where sources or destinations are marked as excluded.
- Inhibit Action - inhibit all the crosspoints between the sources and destinations that satisfy the crosspoint type and priority criteria except where sources or destinations are marked as excluded.
- Isolate Action - isolate all the crosspoints between the sources and destinations that satisfy the crosspoint type and priority criteria except where sources or destinations are marked as excluded. When isolate actions are applied to bidirectional crosspoints it will only isolate the source to destination part of the audio path, not the destination to source part.
- IFB Action – Enables an IFB as a crosspoint action.

If you select IFB Action, you can right-click Source Action to select the type of IFB Action you require:

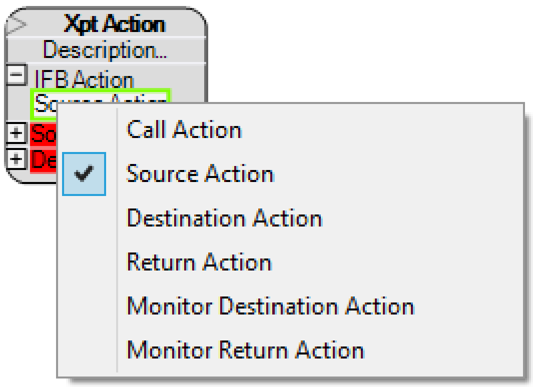


**Figure 1-60 IFB Source Action list**

Choose from the following source actions:

**Call Action** – Temporarily adds a caller to the IFB. If deleted, the caller is not reapplied when the action is removed.

**Source Action** – Temporarily adds a source to the IFB.

**Destination Action** - Temporarily adds a destination to the IFB.

**Return Action** - Temporarily adds a return to the IFB.

**Monitor Destination Action** - Temporarily adds a monitor destination to the IFB. This is also known as a destination listen.

**Monitor Return Action** - Temporarily adds a monitor return to the IFB. This is also known as an IFB listen.

## Crosspoint Type

Open the crosspoint action menu and right click on the current crosspoint type to display the menu of crosspoint types.

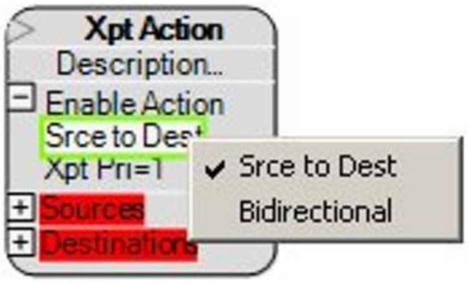


**Figure 1-61: Crosspoint Type List**

The crosspoint action can be set to operate when either source to destination crosspoints are made or bidirectional crosspoints are made between any of the sources and destinations configured. Right click on the menu item to select the crosspoint type.

## Crosspoint Priority

The crosspoint priority defines the priority at which the action is applied to the crosspoints. For a crosspoint action to change the state of a crosspoint it must be set to a priority higher than the crosspoint.

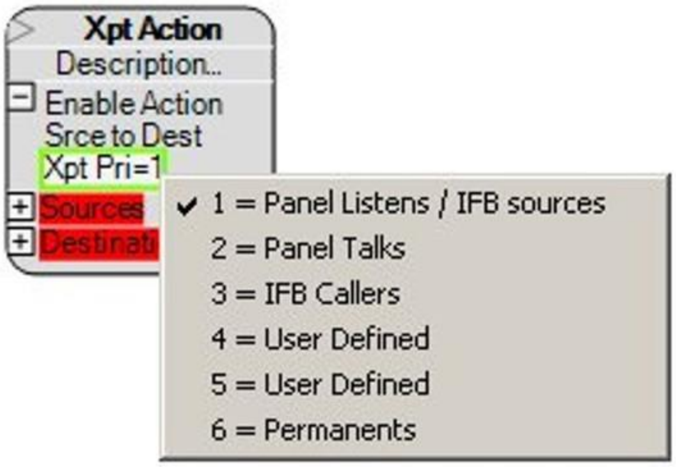


**Figure 1-62: Crosspoint Action Priority**

For example, to override panel talk crosspoints at priority two with a crosspoint action the action priority must be set to three or higher.

## Crosspoint Action Sources

Crosspoint action sources can be added to the list by dragging and dropping devices from the Direct/Interfaces, Fixed Groups and Panels lists onto the source list whether or not it is open. If the source list is opened then dropping a new source onto an existing source will replace it. If there are no items already assigned to the source list then the list name will be highlighted in red. If there are items already assigned the list will not be highlighted but instead will be surrounded by a green box.

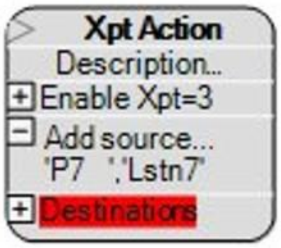


**Figure 1-63: Crosspoint Action Source List**

When the menu name is highlighted in yellow the item can be dropped onto the list.

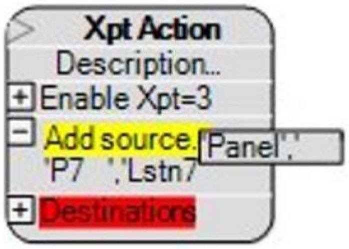


**Figure 1-64: Adding a New Source**

Right-clicking on 'Add source...' will display a menu allowing all the ports or all the panels in the configuration to be added to the source list.

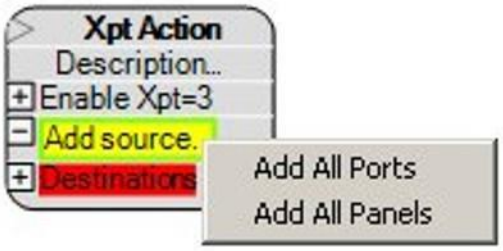


**Figure 1-64: Adding All Ports or Panels to Crosspoint Action Source**

Sources in the list can be copied, cut, deleted or excluded by selecting the required items from the list and right clicking to display the options list. Multiple items on the list can be selected by holding down the Shift key while selecting items.

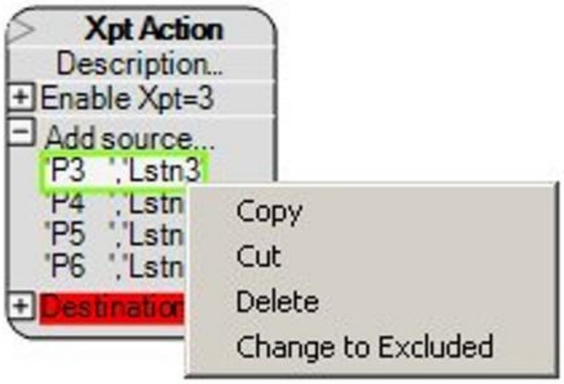


**Figure 1-65: Crosspoint Action Source Options**

Items that are cut or copied can be pasted into other source or destination lists. Deleting an item removes it from the list while the 'Change to Excluded' option allows a source to be excluded from consideration when acting on crosspoints. Any source that has been excluded is shown in red. If 'All Ports' is present in the source list this cannot be excluded. To re-include a destination that has been excluded select it and right click to open the actions menu and select 'Change to Included'.

## Crosspoint Action Destinations

Crosspoint action destinations can be added to the list by dragging and dropping devices from the Direct/Interfaces, Fixed Groups and Panels lists onto the destination list whether or not it is open. If the destination list is opened then dropping a new destination onto an existing destination will replace it. If there are no items already assigned to the destination list then the list name will be highlighted in red. If there are items already assigned the list will not be highlighted but instead will be surrounded by a green box. In the case of IFB actions, IFBs and fixed groups, if a fixed group s added to an IFB crosspoint action as a destination only the IFB in the group will be actioned.

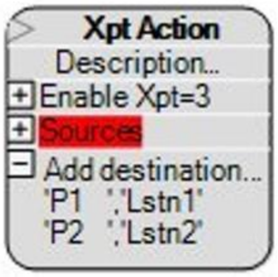


**Figure 1-66: Destination Menu Selected**

When the list name is highlighted in yellow the item can be dropped into the list.

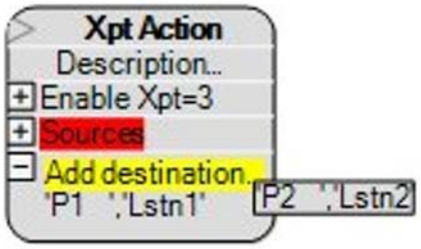


**Figure 1-67: New Destination Item Added**

Right-clicking on 'Add destination...' will display a menu allowing all the ports or all the panels in the configuration to be added to the destination list.

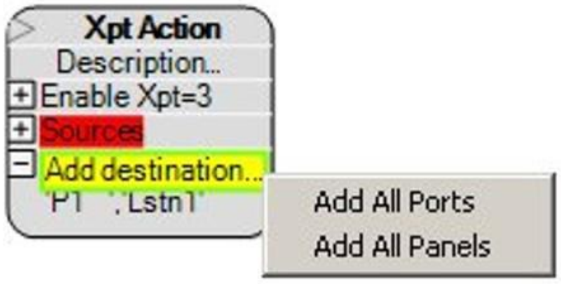


**Figure 1-68: Adding All Ports or Panels to Crosspoint Action**

Destinations in the list can be copied, cut, deleted or excluded by selecting the required items from the list and right clicking to display the options list. Multiple items on the list can be selected by holding down the Shift key while selecting items.

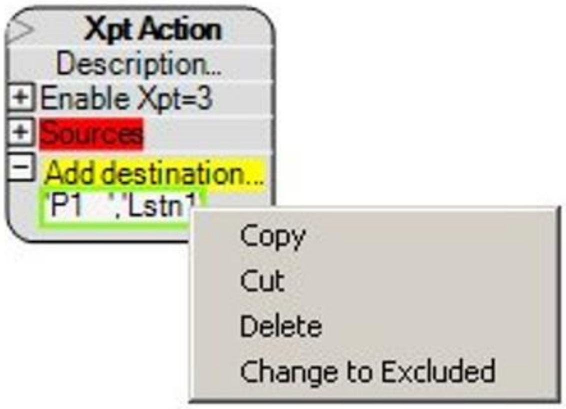


**Figure 1-69: Crosspoint Action Destination Options**

Items that are cut or copied can be pasted into other source or destination lists. Deleting an item removes it from the destination list while the 'Change to Excluded' option allows a destination to be excluded from consideration when triggering an output. Any destination that is excluded is shown in red. If 'All Ports' is present in the destination list this cannot be excluded.

Right clicking on an empty destination list without expanding the list will display an additional option of 'Pin to Source List'.

Right clicking on an empty destination list without expanding the list will display an additional option of 'Pin to Source List'.

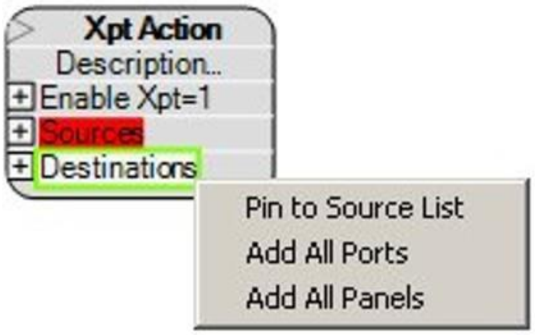


**Figure 1-70: Pin to Source List Destination Option**

Selecting the 'Pin to Source List' option replaces the 'Destinations' menu with 'Dests => Sources'. To reinstate the 'Destinations' menu right click on 'Dests => Sources' and select 'Detach from Source List'.

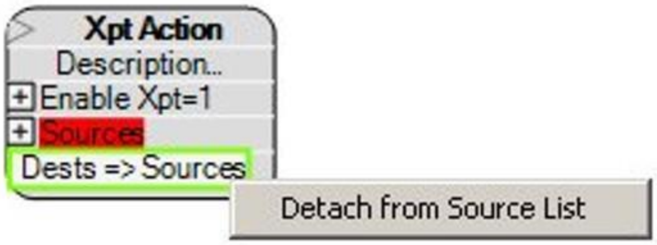


**Figure 1-71: Delete Pin to Source List Option**

The 'Dests => Sources' option replaces the destination list with a matrix of crosspoints between all the sources in the source list. This is shown by the crosspoint options menu being replaced by a new 'All Xpts' menu. Right-clicking on the 'All Xpts' menu will display a list of options allowing the crosspoint matrix to be modified.

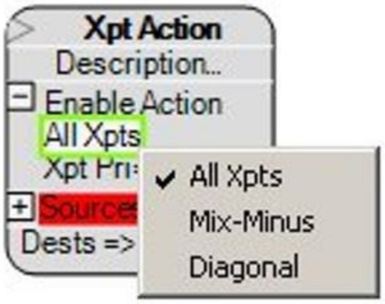


**Figure 1-72: Crosspoint Pin to Source Options**

The crosspoint options for Pin to Source are:

- All Xpts - acts on every crosspoint between sources in the source list. The example below shows the table for sources 1 - 6.

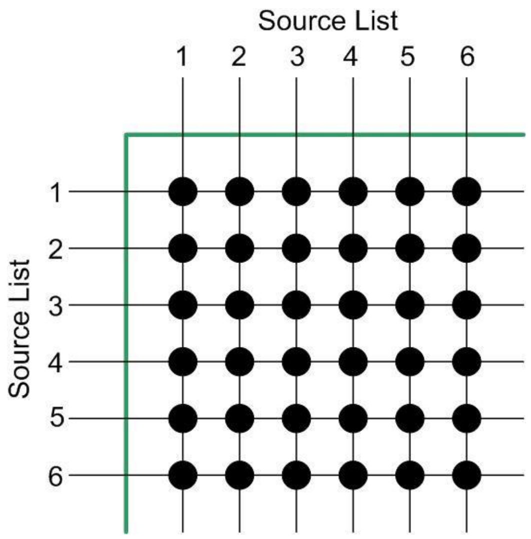


**Figure 1-73: All Possible Crosspoints Set as Action**

- Mix-Minus - acts on every crosspoint between sources on the source list except loopback crosspoints that form the diagonal on the crosspoint matrix. The example below shows the table for sources 1 - 6.

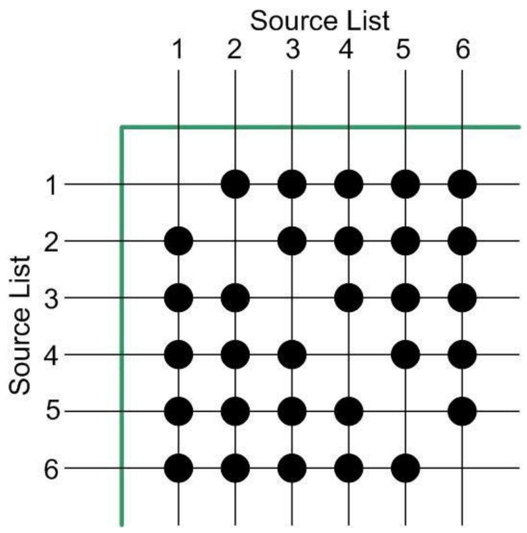


**Figure 1-74: Mix-Minus Crosspoints**

- Diagonal - triggers on all loopback crosspoints .i.e. where sources on the source list are looped back to themselves. The example below shows the table for sources 1 - 6.

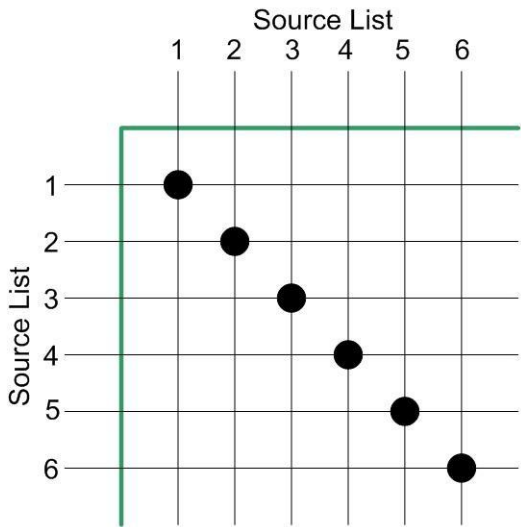


**Figure 1-75: Loopback Crosspoints**

## 1.3.2 LOGIC ELEMENTS

Logic elements are used to perform logical operations on the outputs of control sequence elements and pass the result to the input of other control sequence elements. This allows complex sequences of actions depending on various conditions to be built up and programmed into the matrix system. Right click on the logic elements in the Logic Elements pane to display an options menu. This menu allows the user to display a truth table for the logic element or copy the logic element to the design window.

Logic elements can be inserted existing connections by right-clicking on the connection to display the options menu and selecting 'Insert Gate Type'. A list of logic elements will be displayed for insertion into the connection.

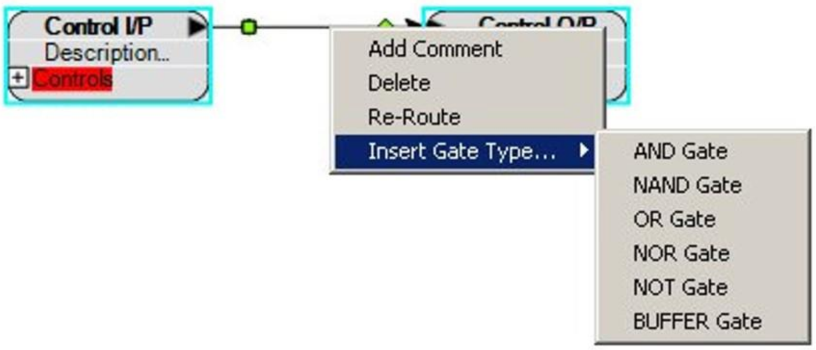


**Figure 1-76: Inserting a Logic Element into a Connection**

The logic elements available are described below.

## AND Gate

Combines two or more inputs to generate a single output. The default is two inputs but by right clicking on the AND gate to display the options menu additional inputs can be added. Unused inputs will default to the TRUE state. The output is only true if all the inputs are true. The AND gate adds a 25ms processing delay.

| Input A | Input B | Output |
|---|---|---|
| False | False | False |
| False | True | False |
| True | False | False |
| True | True | True |

**Table 1-1: Truth Table for AND Logic Element**

Right clicking on the logic element in the design window displays an options menu.

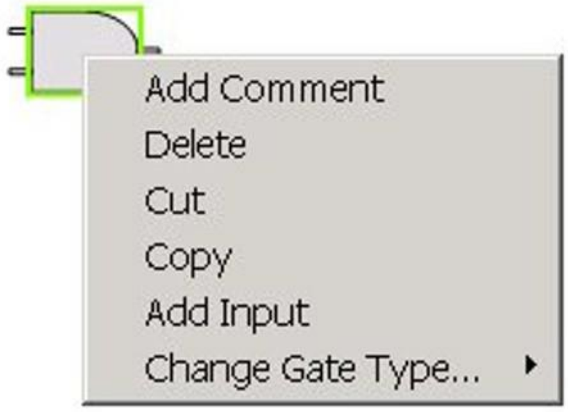


**Figure 1-77: Menu Options for AND Logic Element**

- Add Comment - add a comment to the logic element.
- Delete - delete logic element from design window.
- Cut - cut logic element from design window.
- Copy - copy logic element on design window.
- Add Input - add an input to the logic element.
- Change Gate Type - replace the logic element with one selected from the drop-down list.

## NAND Gate

Combines two or more inputs to generate a single output. The default is two inputs but by right clicking on a NAND gate to display the menu additional inputs can be added. Unused inputs will default to the TRUE state. The output is only true if at least one input is false. The NAND gate adds a 25ms processing delay.

| Input A | Input B | Output |
|---|---|---|
| False | False | True |
| False | True | True |
| True | False | True |
| True | True | False |

**Table 1-2: Truth Table for NAND Logic Element**

Right clicking on the logic element in the design window displays an options menu.

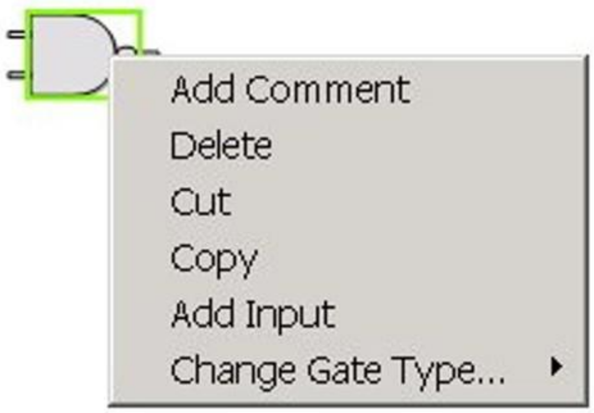


**Figure 1-78: Menu Options for NAND Logic Element**

- Add Comment - add a comment to the logic element.
- Delete - delete logic element from design window.
- Cut - cut logic element from design window.
- Copy - copy logic element on design window.
- Add Input - add an input to the logic element.
- Change Gate Type - replace the logic element with one selected from the drop-down list.

## OR Gate

Combines two or more inputs to generate a single output. The default is two inputs but by right clicking on an OR gate to display the menu additional inputs can be added. Unused inputs will default to the TRUE state. The output is only true if one or more inputs are true. The OR gate adds a 25ms processing delay.

| Input A | Input B | Output |
|---|---|---|
| False | False | False |
| False | True | True |
| True | False | True |
| True | True | True |

**Table 1-3: Truth Table for OR Logic Element**

Right clicking on the logic element in the design window displays an options menu.

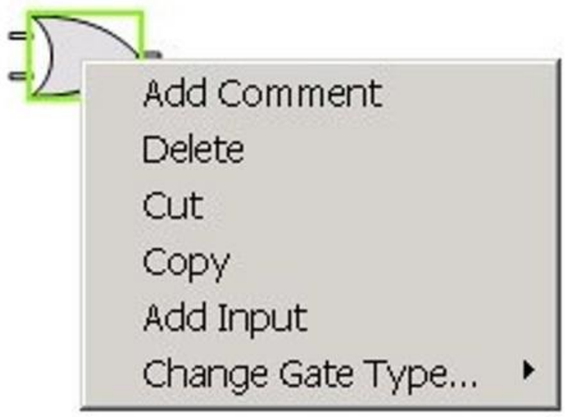


**Figure 1-79: Menu Options for OR Logic Element**

- Add Comment - add a comment to the logic element.
- Delete - delete logic element from design window.
- Cut - cut logic element from design window.
- Copy - copy logic element on design window.
- Add Input - add an input to the logic element.
- Change Gate Type - replace the logic element with one selected from the drop-down list.

## NOR Gate

Combines two or more inputs to generate a single output. The default is two inputs but by right clicking on a NOR gate to display the menu additional inputs can be added. Unused inputs will default to the TRUE state. The output is only true if all inputs are false. The NOR gate adds a 25ms processing delay.

| Input A | Input B | Output |
|---|---|---|
| False | False | True |
| False | True | False |
| True | False | False |
| True | True | False |

**Table 1-4: Truth Table for NOR Logic Element**

Right clicking on the logic element in the design window displays an options menu.

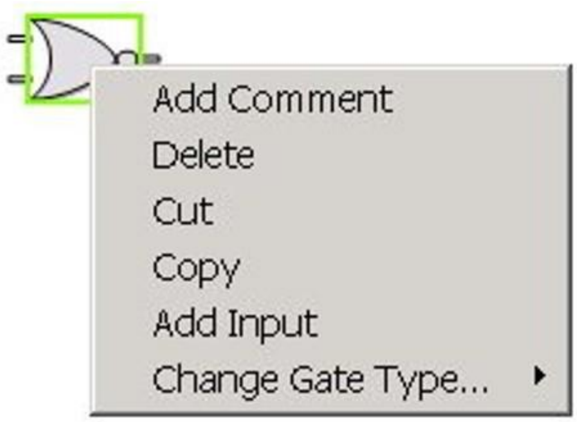


**Figure 1-80: Menu Options for NOR Logic Element**

- Add Comment - add a comment to the logic element.
- Delete - delete logic element from design window.
- Cut - cut logic element from design window.
- Copy - copy logic element on design window.
- Add Input - add an input to the logic element.
- Change Gate Type - replace the logic element with one selected from the drop-down list.

## BUFFER Element

Used between a crosspoint trigger and a crosspoint action to reduce the system resource usage. There is a constraint on the maximum number of possible actions by crosspoint triggers and crosspoint actions imposed by system resources. In general the number of possible triggers times the number of possible actions should not exceed 4095. So if there are 16 possible triggers specified in a crosspoint trigger and 16 possible crosspoint actions specified in a crosspoint action the number of actions would be:

16 triggers x 16 actions = 256 events

which would be acceptable. If the result of setting up a system of crosspoint triggers and crosspoint actions created more than 4095 possible actions an error would be reported when the configuration was downloaded.

In this case a buffer logic element should be placed between the crosspoint trigger and crosspoint action. In this way the number of actions the trigger crosspoint has to make is limited to the number of trigger crosspoints, which only have to trigger the buffer. The buffer will then act on the crosspoints in the crosspoint action.

The BUFFER element adds a 25ms processing delay.

| Input A | Output |
|---|---|
| False | False |
| True | True |

**Table 1-5: Truth Table for BUFFER Logic Element**

Right clicking on the logic element in the design window displays an options menu.

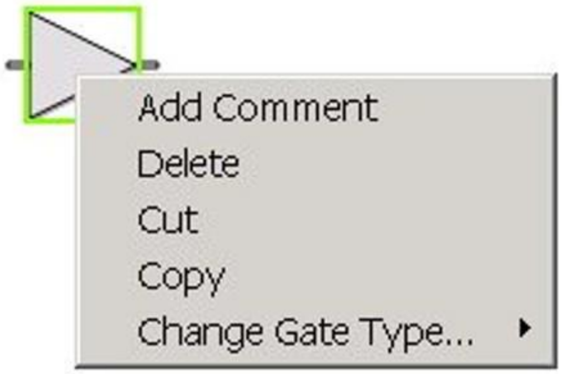


**Figure 1-81: Menu Options for BUFFER Logic Element**

Add Comment - add a comment to the logic element.

Delete - delete logic element from design window.

Cut - cut logic element from design window.

Copy - copy logic element on design window.

Change Gate Type - replace the logic element with one selected from the drop-down list.

## NOT Element

A NOT element inverts the input so that when the input is OFF the output is ON; when the input is ON the output is OFF. The NOT function adds a 25ms processing delay.

| Input A | Output |
|---|---|
| False | True |
| True | False |

**Table 1-6: Truth Table for NOT Logic Element**

Right clicking on the logic element in the design window displays an options menu.

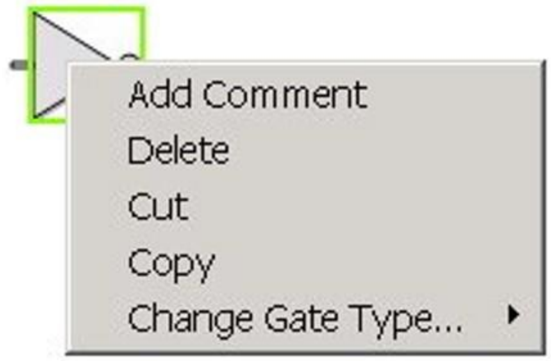


**Figure 1-82: Menu Options for NOT Logic Element**

- Add Comment - add a comment to the logic element.
- Delete - delete logic element from design window.
- Cut - cut logic element from design window.
- Copy - copy logic element on design window.
- Change Gate Type - replace the logic element with one selected from the drop-down list.

## LATCH Element

The latch element creates a true or false output that can be set or cleared by inputs to toggle, set or reset inputs.  The latch element will maintain the state it is set to until that state is changed via a set, reset or toggle.

| Reset | Set | Toggle | Q | /Q |
|---|---|---|---|---|
| True | X | X | False | True |
| False | True | X | True | False |
| False | False | F -> T | /Q | Q |
| X | False | T -> F | Q | /Q |

**Table 1-7: Truth Table for LATCH Logic Element**

Right clicking on the logic element in the design window displays an options menu.

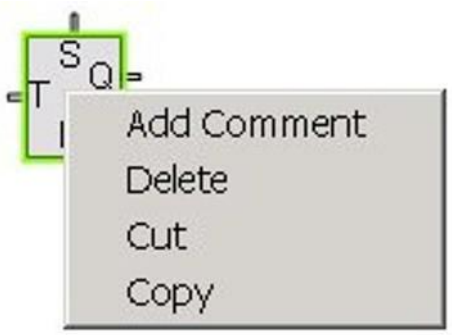


**Figure 1-83: Menu Options for LATCH Logic Element**

- Add Comment - add a comment to the logic element.
- Delete - delete logic element from design window.
- Cut - cut logic element from design window.
- Copy - copy logic element on design window.

The latch element can be used with the toggle input only connected and the set and reset inputs not connected.  In this case the latch will change state when it is toggled by an external input.  The toggle operates on the rising edge of an input so if the input goes to true the latch will toggle to the opposite to its current state.  When the toggle input goes false the latch will remain in its current state until the toggle input goes true again.

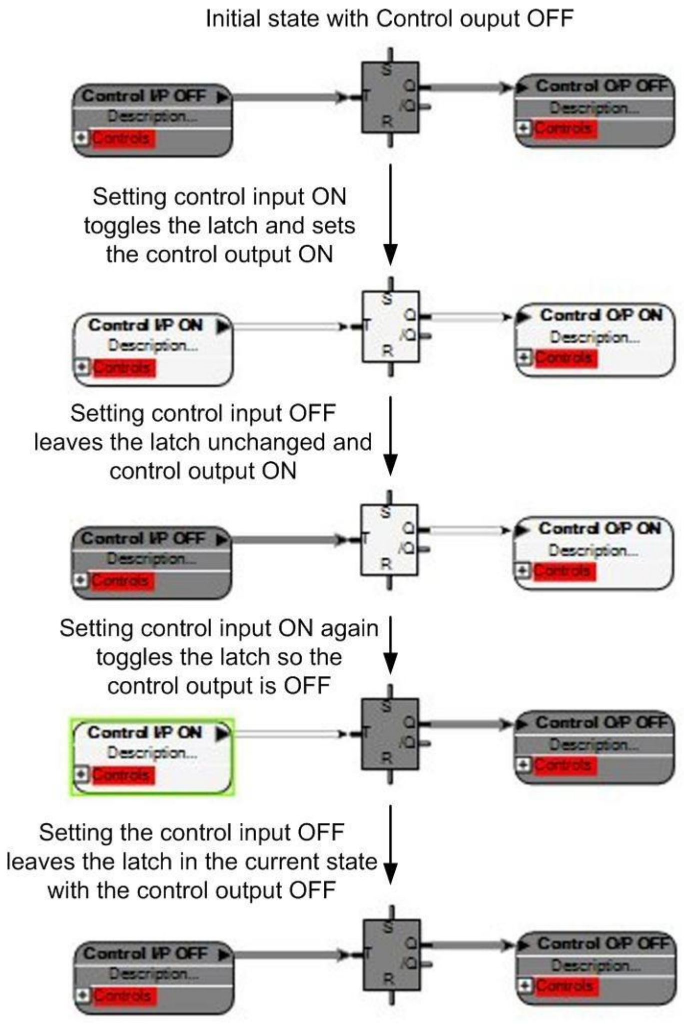

Initial state with Control ouput OFF
Control I/P OFF
Description...
Controls
S
T
Q
/Q
R
Control O/P OFF
Setting control input ON toggles the latch and sets the control output ON
Control I/P ON
Control O/P ON
Setting control input OFF leaves the latch unchanged and control output ON
Setting control input ON again toggles the latch so the control output is OFF
Setting the control input OFF leaves the latch in the current state with the control output OFF

**Figure 1-84: Latch Sequence Using Toggle**

This allows a toggle input to toggle normal and inverted outputs so that input events will enable outputs which then remain enabled until the latch is toggled again or reset.

An example of using the latch with toggle, set and reset connected is shown in Figure 1-85.

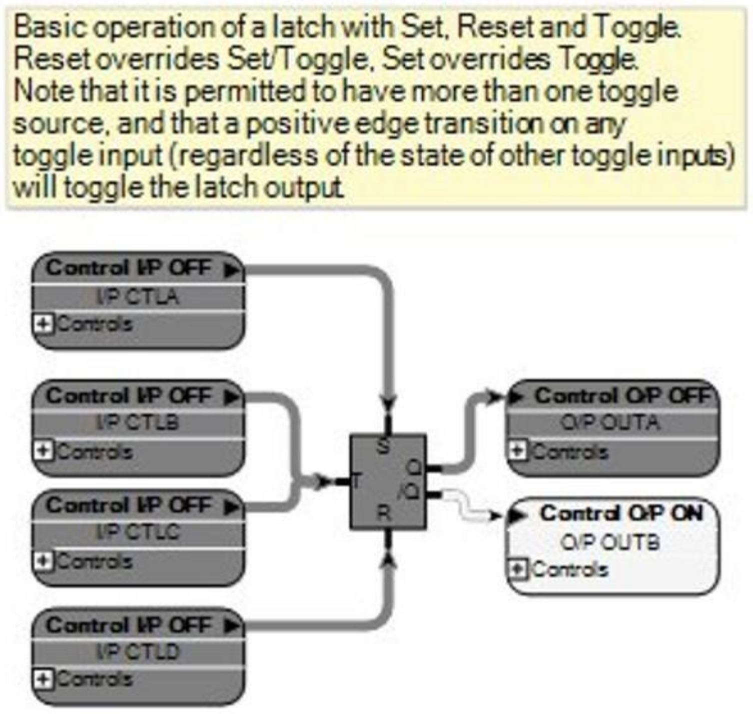

Basic operation of a latch with Set, Reset and Toggle.
Reset overrides Set/Toggle, Set overrides Toggle.
Note that it is permitted to have more than one toggle source, and that a positive edge transition on any toggle input (regardless of the state of other toggle inputs) will toggle the latch output.

**Figure 1-85: Latch Example using All Inputs**

The LATCH function adds a 25ms processing delay.

## ENABLE Element

The ENABLE logic element allows a logic true or always on input to be placed in a control sequence design. This allows control sequences to be created with temporary external inputs or stubs which are known to be always on. This logic element is useful for testing the logic design.

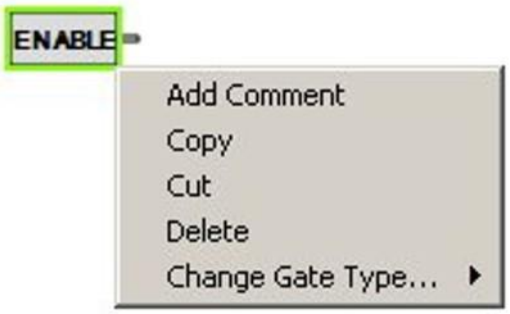


**Figure 1-86: Menu Options for Enable Logic Element**

- Add Comment - add a comment to the logic element.
- Delete - delete logic element from design window.
- Cut - cut logic element from design window.

- Copy - copy logic element on design window.
- Change Gate Type - replace the logic element with one selected from the drop-down list.

## DISABLE Element

The DISABLE logic element allows a logic false or always off input to be placed in a control sequence design. This allows control sequence designs to be created with temporary external inputs or stubs which are known to be always off. This logic element is useful for testing the logic design.

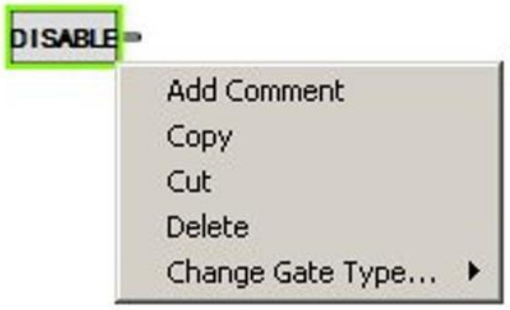


**Figure 1-87: Menu Options for Disable Logic Element**

- Add Comment - add a comment to the logic element.
- Delete - delete logic element from design window.
- Cut - cut logic element from design window.
- Copy - copy logic element on design window.
- Change Gate Type - replace the logic element with one selected from the drop-down list.

Some Example of the use of logic elements are shown in Figure 1-88.

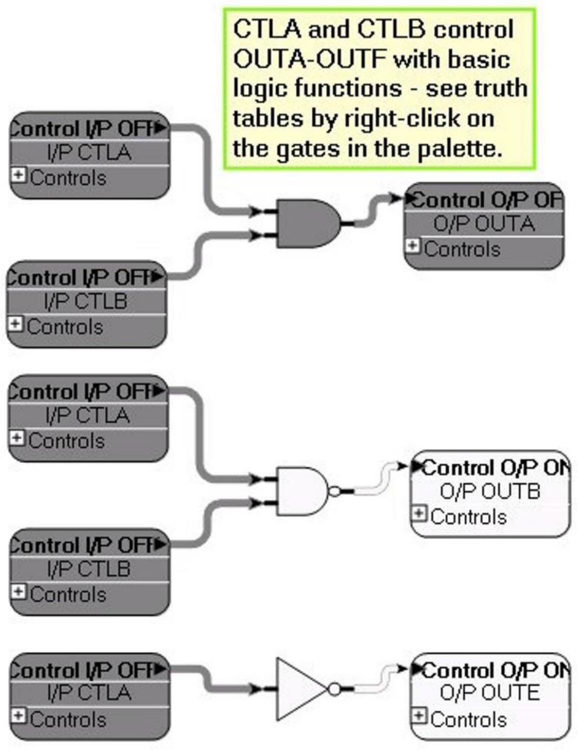


**Figure 1-88: AND, NAND and BUFFER Logic Elements**

# 2 APPENDIX A CONTROL MACRO EDITOR

## 2.1 INTRODUCTION TO CONTROL MACRO EDITOR

Control macros allow the configuration map that controls the matrix operation to be directly modified via control macros. Each control macro contains a series of commands with each defined command representing an action carried out on an object in the configuration. An object may be a port, an input or output device or label.

The main use of control macro scripts is to select controls which have already been configured using the EHX client, and modify the actions that they trigger when activated.

Each defined command is named and can have multiple inputs and outputs and combination logic. These commands take the form of actions to be associated with controls, and the control macro editor will assist the author by providing an overview of available actions and the parameters each requires in order to function.

Conditional logic is available (i.e. with AND, OR logic), with examples and code hints supplied by the control macro editor environment.

Examples of the use of control macros when coupled with EHX Controls and port configuration are:

- To enable or disable a route between any source and a named destination which may be conditional on the status of other Controls, Route based Controls or GPIs.
- To enable or disable a named panel’s loudspeaker (dimming/ muting)
- To remotely enable a named panel’s microphone muting
- To remotely enable a named panel’s headset/ microphone selection
- To remotely enable a named panel’s nominated Key LED signal activation
- To remotely enable a named panel’s nominated Relay control The control macro editor enables the user to:
- Define control macros
- Reference control macros by name
- Assign named macro functions to controls

**Note:** Note: Control Macros are only available to ECS V4.0 or later. The Control Macro Application is a stand-alone application requiring a license key. EHX then imports the macros for use within the EHX environment.

## 2.2 CONTROL MACRO LANGUAGE

The Microsoft .NET Framework is used to provide the scripting facility through the use of dynamic code generation (CodeDOM). This provides the facility to compile control macro into a binary file (an Assembly) rather than the more traditional 'interpreted' control macro of other languages such as VBScript.

Using the .NET Framework as the scripting environment provides the stability and support that the framework has, along with gaining from the .NET Framework features of:

- Managed application environment
- Garbage collector memory management
- Control macros are written in C#

### 2.2.1 EXAMPLE CONTROL MACRO

The following is an example of a control macro created using the control macro editor.

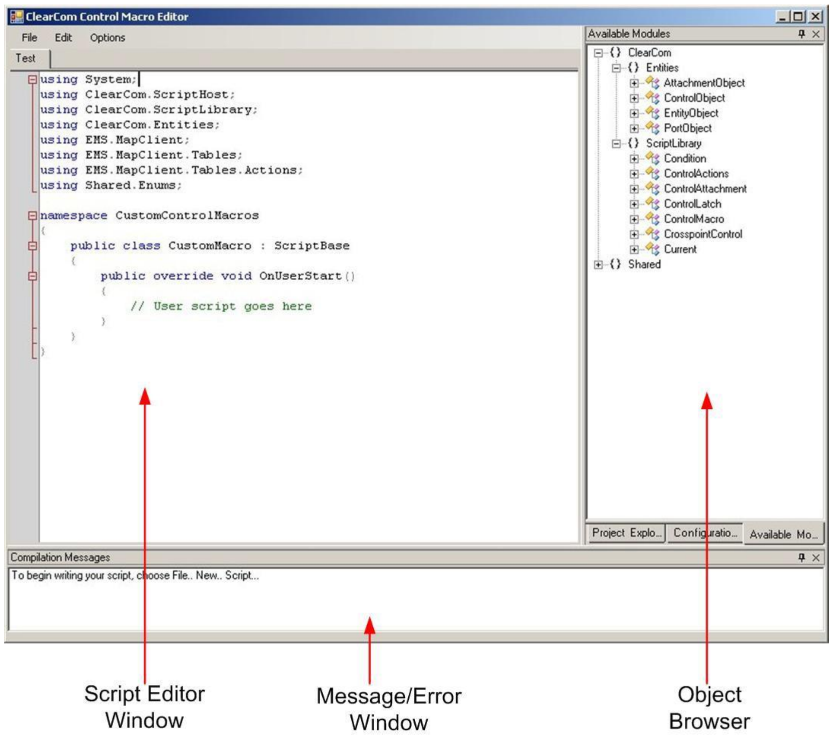
```
Control Macro ExampleScript

using System;                                         //
automatically generated using ClearCom.ScriptHost;

using ClearCom.ScriptLibrary; using
ClearCom.Entities; using EMS.MapClient;

using EMS.MapClient.Tables;

using EMS.MapClient.Tables.Actions; namespace

CustomControlMacros

{
         public class CustomMacro  :  ScriptBase
                            {
                    public override void OnUserStart()
                            {
                       // User Script entered here
Control redLightControl = ExistingControlFromLabel("RDLT");

// gets an already existing Control, set up from the Control Manager within the EHX client
and allows it to be programmed

redLightControl.Triggers(new InhibitRoute(5, 6));

//  When the red light control is fired (studio moves into Live mode) the route between ports 5
and 6 is

// inhibited. The control editor prompts the script author for either a port number or port

name redLightControl.Triggers(new ChangeStatus("DIR", HardwareStatus.LoudspeakerCut);

// changes a large number of panel properties by selecting a panel by name and then triggering
a change

redLightControl.Triggers(new ChangeStatus("DIR", 5);
// Changes LED 5 on panel DIR
                         }
                         }
                                 }
```

## 2.3 CONTROL MACRO EDITOR

To create and edit the Control Macro, a control macro editor is provided. This consists of:

- A main control macro editor window
- An object browser
- A message window

An illustration of the control macro editor is shown below.



```
using System;
using ClearCom.ScriptHost;
using ClearCom.ScriptLibrary;
using ClearCom.Entities;
using EMS.MapClient;
using EMS.MapClient.Tables;
using EMS.MapClient.Tables.Actions;
using Shared.Enums;

namespace CustomControlMacros
{
    public class CustomMacro : ScriptBase
    {
        public override void OnUserStart()
        {
            // User script goes here
        }
    }
}
```

```
To begin writing your script, choose File.. New.. Script...
```

**Figure A-1: Control Macro Editor Screen**

### 2.3.1 CONTROL MACRO EDITOR WINDOW

The control macro editor window provides full access for editing the control macro scripts while also providing assistance to the user in the form of coloured syntax, 'intellisense' (offering context sensitive coding options) and code completion.

### 2.3.2 OBJECT BROWSER

The Object Browser gives a complete display of the objects and logic available to be used to construct control macros. This gives a detailed view of all the contained objects, their constructors, methods and properties. This view will be generated using the powerful reflection capability that is part of the .NET Framework.

### 2.3.3 MESSAGE WINDOW

The Message Window will provide feedback to the user of any validation issues when parsing the control macro. These issues will be flagged as either warnings or errors.

### 2.3.4 RUNNING CONTROL MACROS

Control macros are run at download time and follow a two stage approach of validation and building of the control macro.

The validation stage checks the control macro for warnings or errors which will be reported back to the user. Any errors will prevent the control macro from being compiled.

When the control macro has passed the validation stage, the control macro will be compiled into an Assembly using the Microsoft .NET Framework code compiler. This Assembly is then cached and will only be refreshed if the control macro itself is changed. It is then run at download time, with the output (usually the addition of rack-specific map objects) being sent to the frame together with the EHX-derived configuration.

### 2.3.5 STARTING THE CONTROL MACRO EDITOR

The control macro editor is accessed from Logic by clicking on the 'New' button and selecting 'Control Macro' from the drop-down 'Type' menu.

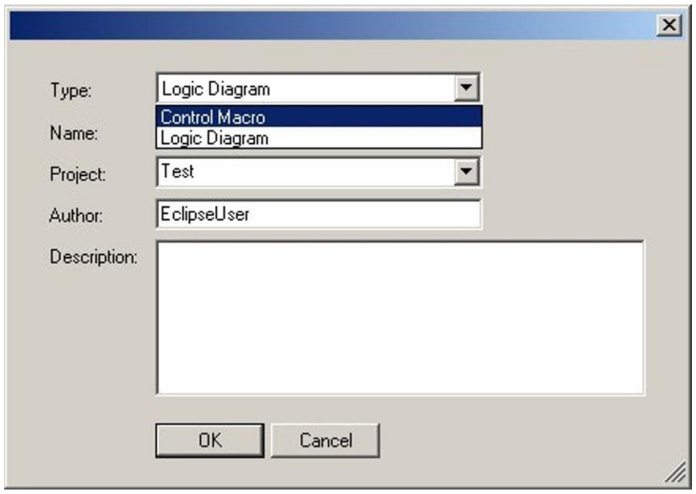

Type: Logic Diagram
Control Macro
Logic Diagram
Name:
Project: Test
Author: EclipseUser
Description:

**Figure A-2: Control Macro Editor from Logic**

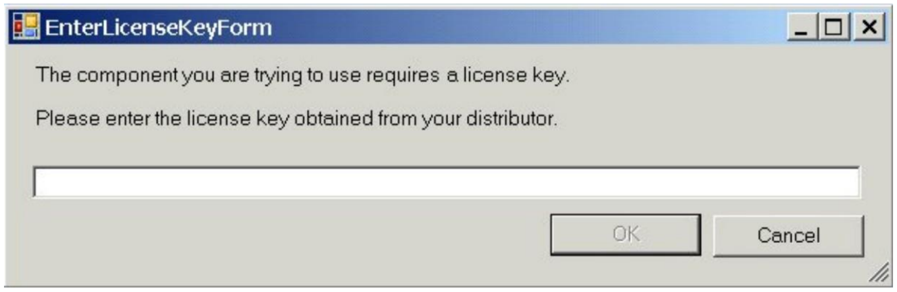
After entering the required fields click on the 'OK' button to open the control macro editor.

Saved control macro files have the same file extension of .ccm as logic files and will be listed with logic files. If the 'Edit

Logic' link is selected for a control macro file the control macro editor will be started automatically rather than the logic diagram editor.

The control macro editor can also be started using a desktop shortcut to the executable if required but this useage is not recommended.

The Eclipse Macro facility is a licensable option and a license key is required to use the editor to create new control macros. When the editor is first started it will request a license key if one has not already been input.


EnterLicenseKeyForm

The component you are trying to use requires a license key.

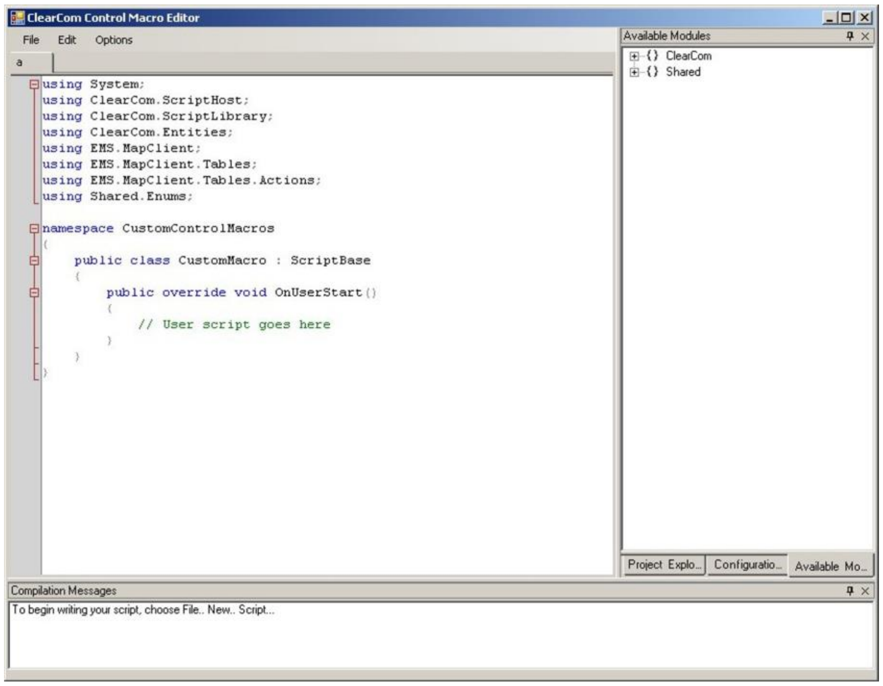
Please enter the license key obtained from your distributor.

**Figure A-3: License Key Request**

Enter the license key obtained from the supplier or distributor and click on the 'OK' button to continue and start the control macro editor. If a valid license key is not entered the control macro editor will exit immediately.

**Note:** Note: When running under Windows Vista the user must have administrator rights in order to enter the control macro editor license key.

When the editor is started from Logic it will display the three windows ready to start a new control macro (if started using the 'New' button) or load an existing macro.



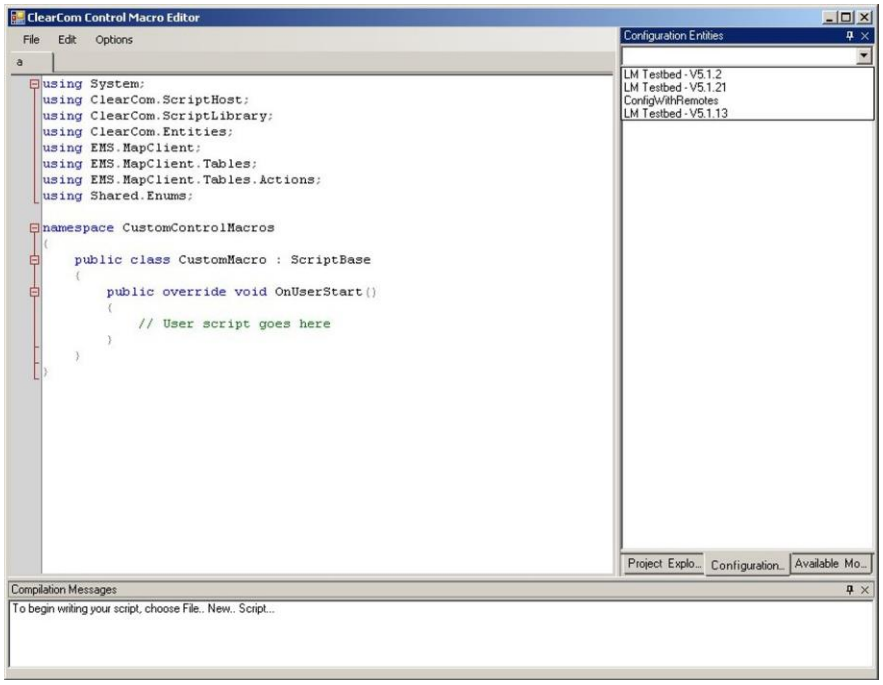
```
using System;
using ClearCom.ScriptHost;
using ClearCom.ScriptLibrary;
using ClearCom.Entities;
using EMS.MapClient;
using EMS.MapClient.Tables;
using EMS.MapClient.Tables.Actions;
using Shared.Enums;

namespace CustomControlMacros
{
    public class CustomMacro : ScriptBase
    {
        public override void OnUserStart()
        {
            // User script goes here
        }
    }
}
```

To begin writing your script, choose File.. New.. Script...

**Figure A-4: Initial Macro Control Macro Editor Display**

## 2.4 CONFIGURATION ENTITIES

Click on the 'Configuration Entities' tab of the object browser to select the system configuration that is to be used by the control macro editor. A drop down menu of all the available system configurations is displayed below the window title.



```
using System;
using ClearCom.ScriptHost;
using ClearCom.ScriptLibrary;
using ClearCom.Entities;
using EMS.MapClient;
using EMS.MapClient.Tables;
using EMS.MapClient.Tables.Actions;
using Shared.Enums;

namespace CustomControlMacros
{
    public class CustomMacro : ScriptBase
    {
        public override void OnUserStart()
        {
            // User script goes here
        }
    }
}
```

```
To begin writing your script, choose File.. New.. Script...
```

**Figure A-5: Configuration Selection**

After a system configuration has been selected the entities that exist in that configuration are displayed in the object browser window under the headings:

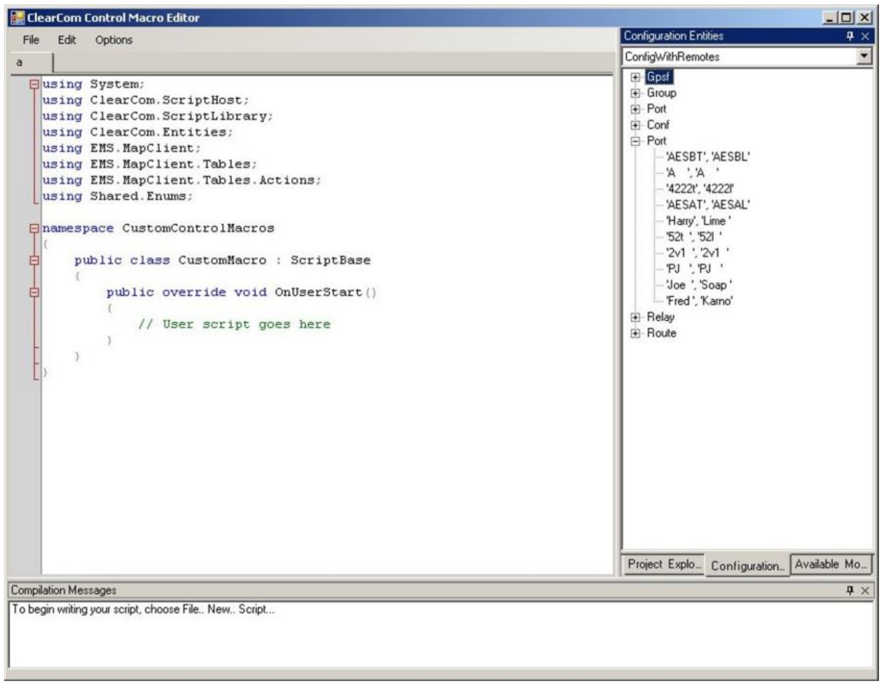
- Gpsf - General Purpose Specific Functions
- Group - fixed groups and sort groups defined in EHX
- Port - system ports defined as Direct in EHX
- Conf - party lines (conferences) defined in EHX
- Port - system ports defined as panels in EHX
- Relay - relays that can be set open or closed
- Route - routes between panels defined in EHX

Each item can be opened to display a list of all the entities of this type in the currently selected system configuration. If the configuration does not include any entities of a type the heading for that entity type is not displayed.

If a new EHX element is made while the control macro editor is opened, then:

1) In EHX Save the configuration(s).

2) Re-select the configuration(s) from the configuration task in order to force the macro editor to refresh its copy of the configuration(s).



```
using System;
using ClearCom.ScriptHost;
using ClearCom.ScriptLibrary;
using ClearCom.Entities;
using EMS.MapClient;
using EMS.MapClient.Tables;
using EMS.MapClient.Tables.Actions;
using Shared.Enums;

namespace CustomControlMacros
{
    public class CustomMacro : ScriptBase
    {
        public override void OnUserStart()
        {
            // User script goes here
        }
    }
}
```

```
To begin writing your script, choose File.. New.. Script...
```

**Figure A-6: Configuration Entities List**

These entities contained in the system configuration selected may be referenced in the control macro as required but the control macro will be specific to that system configuration and should not be used with any other system configuration as it may fail or produce unexpected results.

## 2.5 AVAILABLE MODULES

Click on the 'Available Modules' tab in the object browser to display the menus for the objects used to create the macros. These are divided into the 'ClearCom' modules to construct programs to modify the map and 'Shared' to provide logging and debug capability.

### 2.5.1 CLEARCOM

Click on the 'ClearCom' item and expand the menus to show the object classes available.

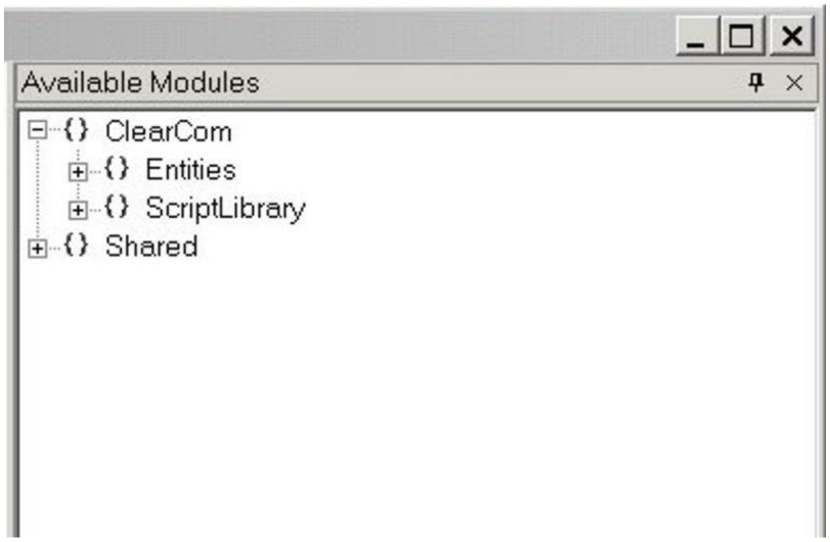


**Figure A-7: ClearCom Module Libraries**

## Entities

The entities section is divided into Attachment Objects which are associated with components, Control Objects that act on system components, Entity Objects that act on the state of system components and Port Objects that act on system ports.

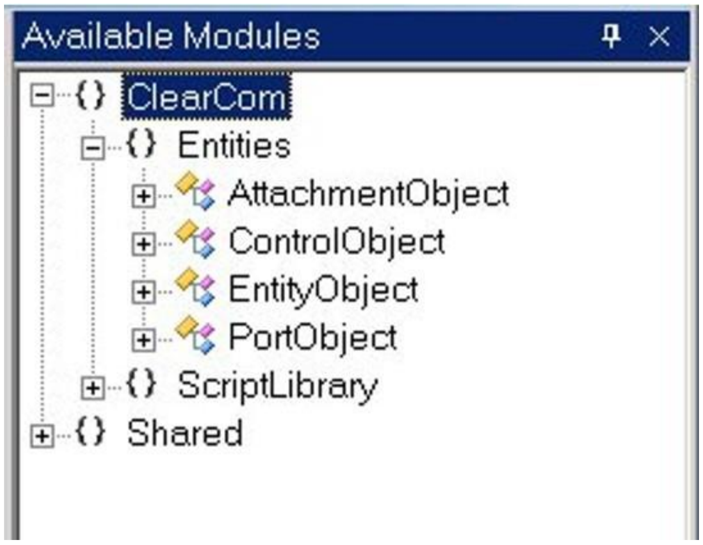
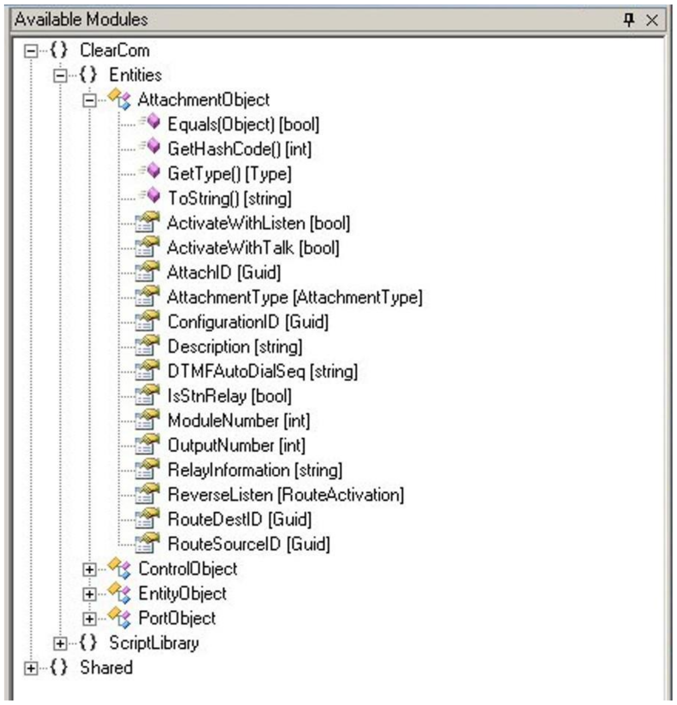


**Figure A-8: Entity Libraries**

## Attachment Objects

When the attachment objects item is selected the list will be expanded to display the attachment objects available and the logic operations that may be used with attachment objects. Attachment objects are attached to components to set or get the properties of those components such as parameters.

Examples of attachment objects are relays, routes and speed dials.

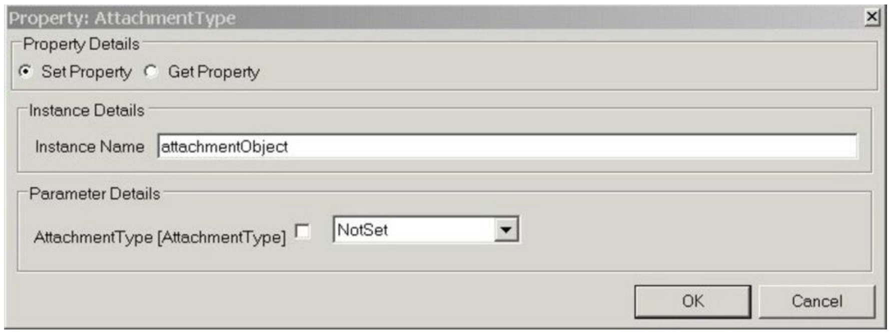


```
Available Modules
⊟ {} ClearCom
  ⊟ {} Entities
    ⊟ AttachmentObject
        Equals(Object) [bool]
        GetHashCode() [int]
        GetType() [Type]
        ToString() [string]
        ActivateWithListen [bool]
        ActivateWithTalk [bool]
        AttachID [Guid]
        AttachmentType [AttachmentType]
        ConfigurationID [Guid]
        Description [string]
        DTMFAutoDialSeq [string]
        IsStnRelay [bool]
        ModuleNumber [int]
        OutputNumber [int]
        RelayInformation [string]
        ReverseListen [RouteActivation]
        RouteDestID [Guid]
        RouteSourceID [Guid]
    ⊞ ControlObject
    ⊞ EntityObject
    ⊞ PortObject
  ⊞ {} ScriptLibrary
⊞ {} Shared
```

**Figure A-9: Attachment Objects Library**

To use an attachment object select the required object by right clicking over it and then dragging it over to the edit window and dropping it in the required position.

When an attachment object is dropped into the control macro the editor will prompt for information such as whether the object is to set or get the component parameter and depending on this any other information that is required such as parameters and how to return the information.



Property: AttachmentType

Property Details: [✓ Set Property] Get Property

Instance Details

Instance Name: attachmentObject

Parameter Details

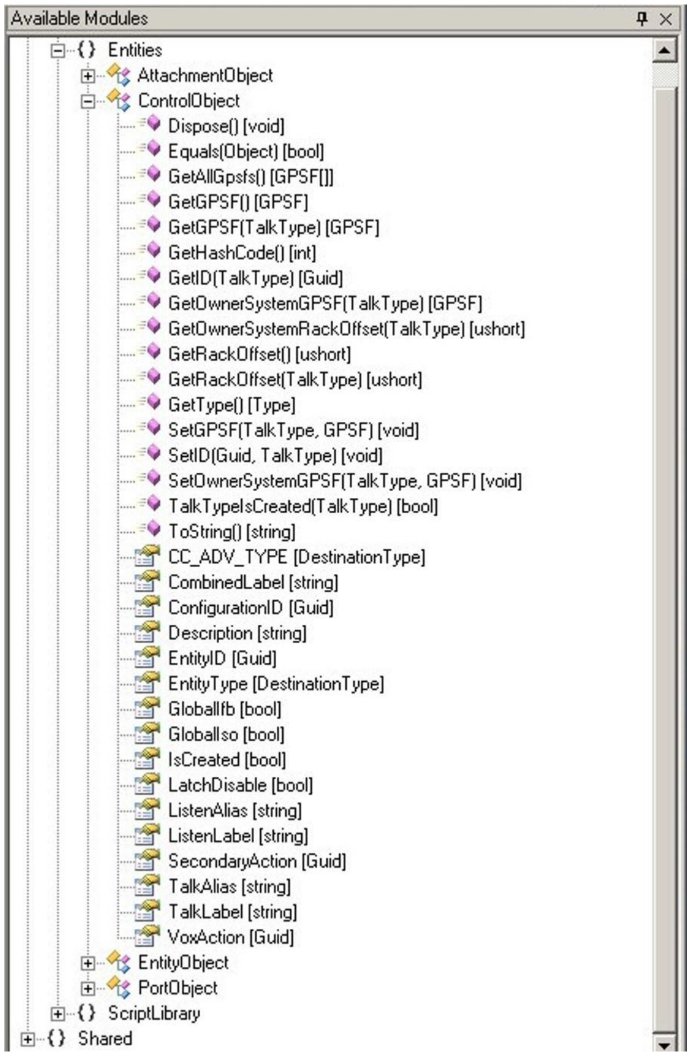
AttachmentType [AttachmentType] ☐ NotSet

**Figure A-10: Example of Attachment Object Properties**

## Control Objects

Control objects act on the components to change their properties in some way. When a control object is dropped into the editor window the editor will prompt for the required settings and parameters for that object.

Control objects are controls created in EHX using the Control Manager function accessed from the Setup Eclipse menu.



```
Available Modules
    { } Entities
        AttachmentObject
        ControlObject
            Dispose() [void]
            Equals(Object) [bool]
            GetAllGpsfs() [GPSF[]]
            GetGPSF() [GPSF]
            GetGPSF(TalkType) [GPSF]
            GetHashCode() [int]
            GetID(TalkType) [Guid]
            GetOwnerSystemGPSF(TalkType) [GPSF]
            GetOwnerSystemRackOffset(TalkType) [ushort]
            GetRackOffset() [ushort]
            GetRackOffset(TalkType) [ushort]
            GetType() [Type]
            SetGPSF(TalkType, GPSF) [void]
            SetID(Guid, TalkType) [void]
            SetOwnerSystemGPSF(TalkType, GPSF) [void]
            TalkTypeIsCreated(TalkType) [bool]
            ToString() [string]
            CC_ADV_TYPE [DestinationType]
            CombinedLabel [string]
            ConfigurationID [Guid]
            Description [string]
            EntityID [Guid]
            EntityType [DestinationType]
            GlobalIfb [bool]
            GlobalIso [bool]
            IsCreated [bool]
            LatchDisable [bool]
            ListenAlias [string]
            ListenLabel [string]
            SecondaryAction [Guid]
            TalkAlias [string]
            TalkLabel [string]
            VoxAction [Guid]
        EntityObject
        PortObject
    { } ScriptLibrary
{ } Shared
```

**Figure A-11: Control Objects List**

An example of the use of a control object is:

> HSON.Triggers(ControlActions.CutLoudspeaker(D4222)); where CutLoudspeaker is the control.

## Entity Objects

Entity objects act on the components to change their state in some way. When an entity object is dropped into the editor window the editor will prompt for the required settings and parameters for that object.

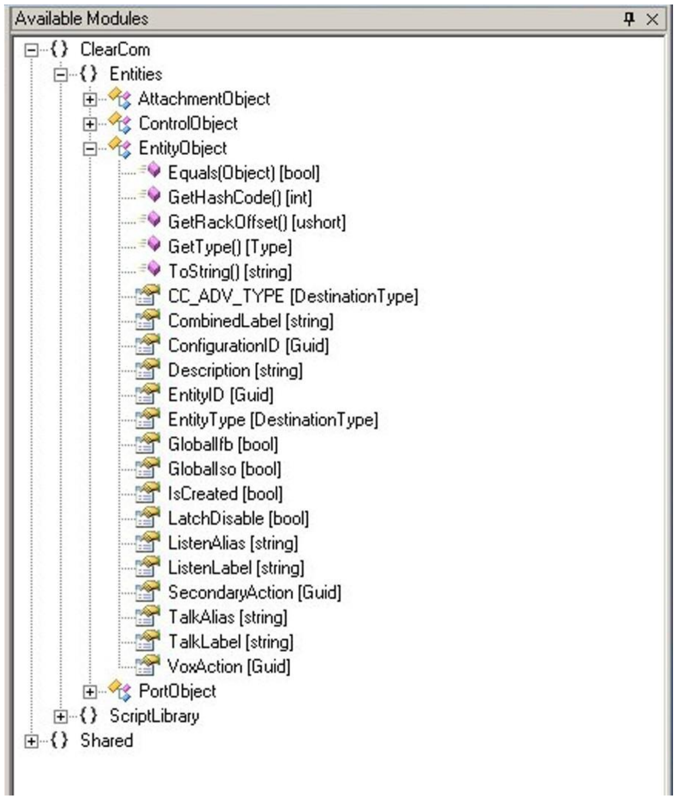


**Figure A-12: Entity Object List**

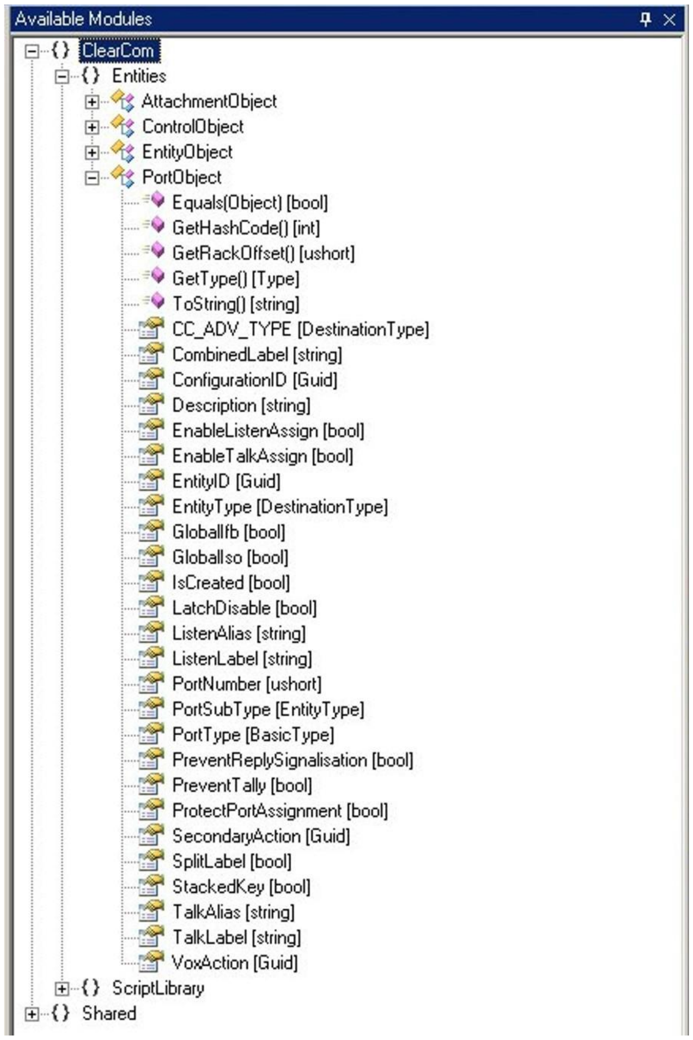
An example of the use of an entity object is:

> HSON.Triggers(ControlActions.CutLoudspeaker(D4222)); where CutLoudspeaker is the control.

## Port Objects

Port objects are used to get information on a system port to change the properties of a system port. When a port object is dropped into the the edit window the editor will prompt for the required settings and parameters for that object and action.

Port objects are normally ports on the system.



```
Available Modules
⊟ { } ClearCom
  ⊟ { } Entities
    ⊞ AttachmentObject
    ⊞ ControlObject
    ⊞ EntityObject
    ⊟ PortObject
        Equals(Object) [bool]
        GetHashCode() [int]
        GetRackOffset() [ushort]
        GetType() [Type]
        ToString() [string]
        CC_ADV_TYPE [DestinationType]
        CombinedLabel [string]
        ConfigurationID [Guid]
        Description [string]
        EnableListenAssign [bool]
        EnableTalkAssign [bool]
        EntityID [Guid]
        EntityType [DestinationType]
        Globallfb [bool]
        Globallso [bool]
        IsCreated [bool]
        LatchDisable [bool]
        ListenAlias [string]
        ListenLabel [string]
        PortNumber [ushort]
        PortSubType [EntityType]
        PortType [BasicType]
        PreventReplySignalisation [bool]
        PreventTally [bool]
        ProtectPortAssignment [bool]
        SecondaryAction [Guid]
        SplitLabel [bool]
        StackedKey [bool]
        TalkAlias [string]
        TalkLabel [string]
        VoxAction [Guid]
  ⊞ { } ScriptLibrary
⊞ { } Shared
```

**Figure A-13: Port Object List**

An example of port object use is:

> PortObject D4222 = ControlMacro.GetPort("D4222"); where D4222 is the port object defined is Matrix Hardware.

## Scriptlibrary

The scriptlibrary section is divided into Conditions which allow components and component parameters to be tested, Control Actions which specify actions to be carried out on system components, Control Attachments which specify actions to be carried out on objects and Control Macros which act on system components.

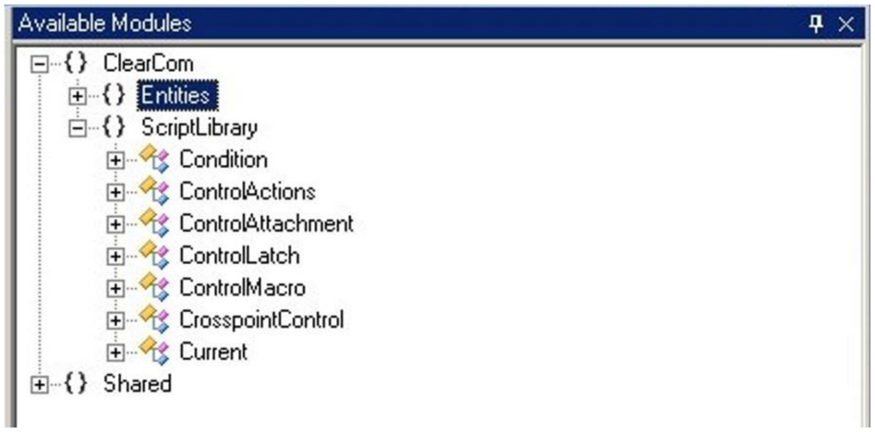


**Figure A-14: Script Library Categories**

## Condition

The condition objects allow the value or state of component parameters to be tested, compared or converted from one format to another.  Conditions are AND and OR.

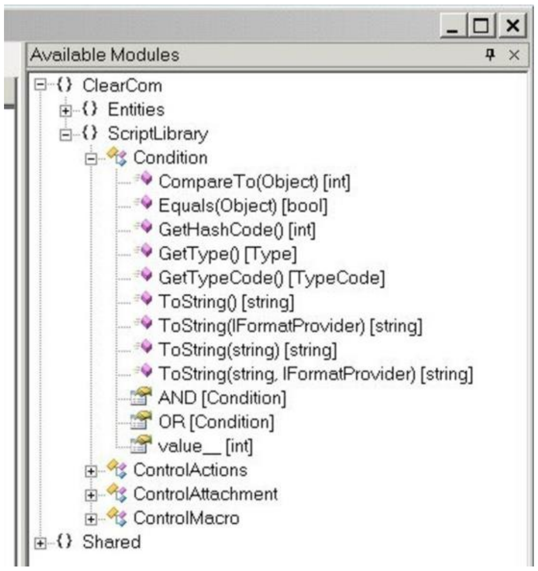


**Figure A-15: Conditions List**

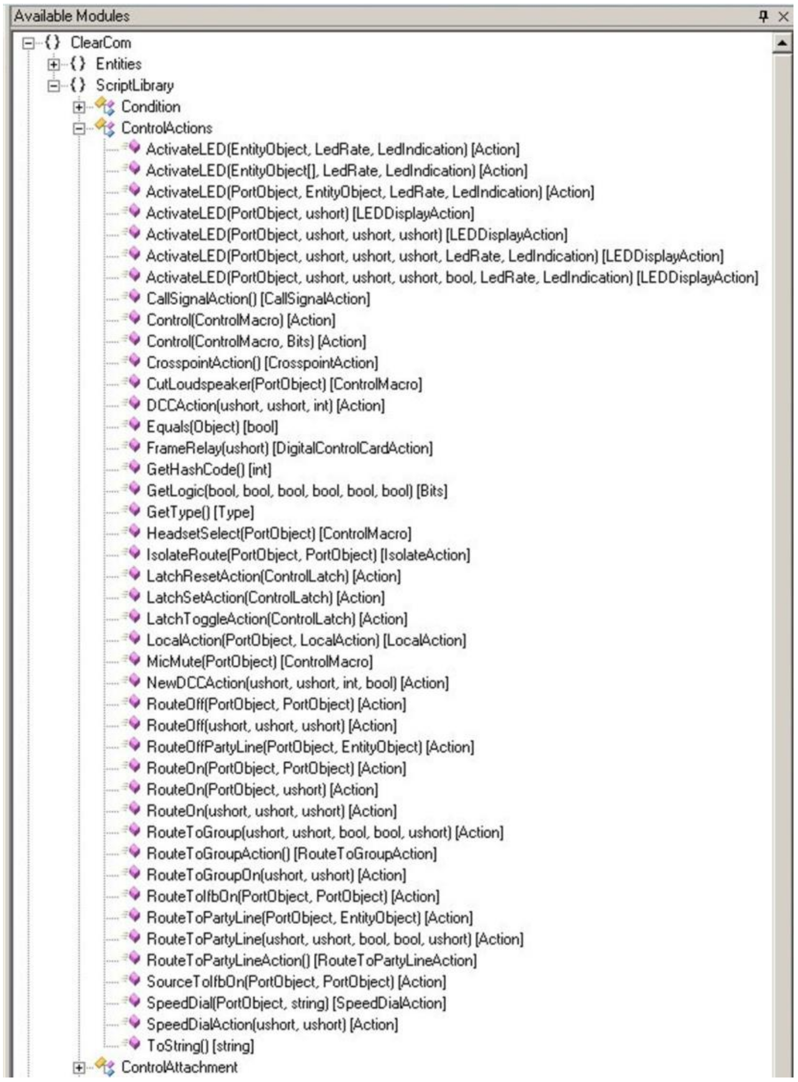
An example of the use of a condition is:

```
FRLY2.TriggersIf(crosspointControl,Condition.AND,AND1);
```

where control FRLY2 is triggered if the elements crosspointControl and AND1 are both true.

## Control Actions

Control actions allow the states of system components such as LEDs, actions (for example when a key is pressed) and routes to be changed for new actions and routes to be created.

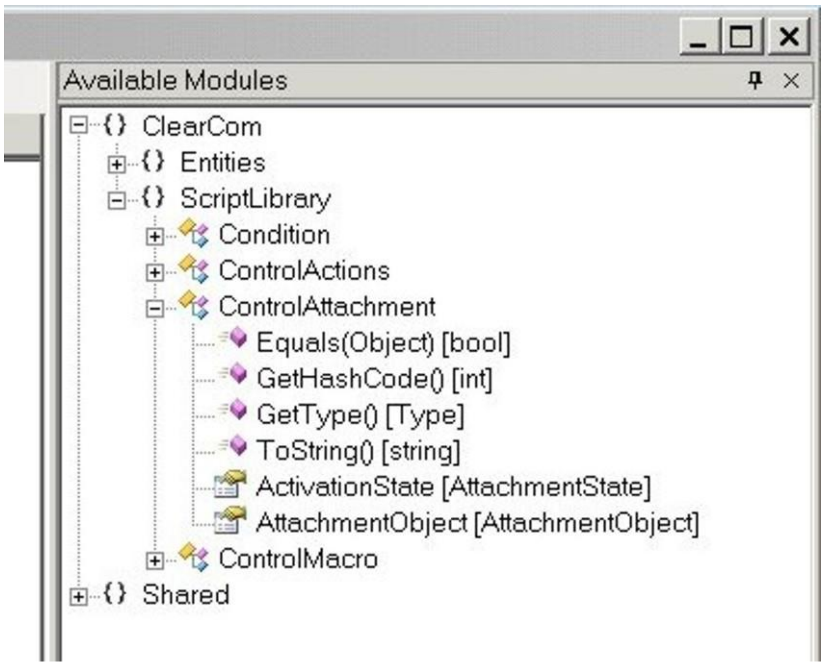


```
Available Modules
ClearCom
  Entities
  ScriptLibrary
    Condition
    ControlActions
      ActivateLED(EntityObject, LedRate, LedIndication) [Action]
      ActivateLED(EntityObject[], LedRate, LedIndication) [Action]
      ActivateLED(PortObject, EntityObject, LedRate, LedIndication) [Action]
      ActivateLED(PortObject, ushort) [LEDDisplayAction]
      ActivateLED(PortObject, ushort, ushort, ushort) [LEDDisplayAction]
      ActivateLED(PortObject, ushort, ushort, ushort, LedRate, LedIndication) [LEDDisplayAction]
      ActivateLED(PortObject, ushort, ushort, ushort, bool, LedRate, LedIndication) [LEDDisplayAction]
      CallSignalAction() [CallSignalAction]
      Control(ControlMacro) [Action]
      Control(ControlMacro, Bits) [Action]
      CrosspointAction() [CrosspointAction]
      CutLoudspeaker(PortObject) [ControlMacro]
      DCCAction(ushort, ushort, int) [Action]
      Equals(Object) [bool]
      FrameRelay(ushort) [DigitalControlCardAction]
      GetHashCode() [int]
      GetLogic(bool, bool, bool, bool, bool, bool) [Bits]
      GetType() [Type]
      HeadsetSelect(PortObject) [ControlMacro]
      IsolateRoute(PortObject, PortObject) [IsolateAction]
      LatchResetAction(ControlLatch) [Action]
      LatchSetAction(ControlLatch) [Action]
      LatchToggleAction(ControlLatch) [Action]
      LocalAction(PortObject, LocalAction) [LocalAction]
      MicMute(PortObject) [ControlMacro]
      NewDCCAction(ushort, ushort, int, bool) [Action]
      RouteOff(PortObject, PortObject) [Action]
      RouteOff(ushort, ushort, ushort) [Action]
      RouteOffPartyLine(PortObject, EntityObject) [Action]
      RouteOn(PortObject, PortObject) [Action]
      RouteOn(PortObject, ushort) [Action]
      RouteOn(ushort, ushort, ushort) [Action]
      RouteToGroup(ushort, ushort, bool, bool, ushort) [Action]
      RouteToGroupAction() [RouteToGroupAction]
      RouteToGroupOn(ushort, ushort) [Action]
      RouteToIfbOn(PortObject, PortObject) [Action]
      RouteToPartyLine(PortObject, EntityObject) [Action]
      RouteToPartyLine(ushort, ushort, bool, bool, ushort) [Action]
      RouteToPartyLineAction() [RouteToPartyLineAction]
      SourceToIfbOn(PortObject, PortObject) [Action]
      SpeedDial(PortObject, string) [SpeedDialAction]
      SpeedDialAction(ushort, ushort) [Action]
      ToString() [string]
    ControlAttachment
```

**Figure A-16: Control Actions List**

## Control Attachments

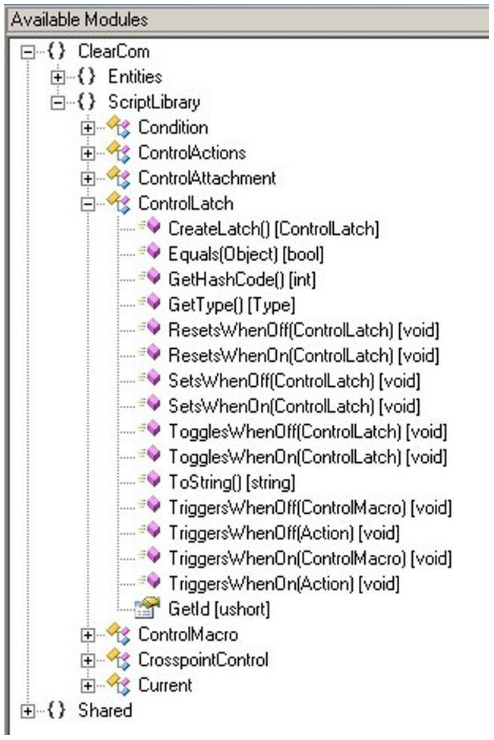
Control attachment objects allow the states of pre-existing system components to be changed.



```
⊟ {} ClearCom
  ⊞ {} Entities
  ⊟ {} ScriptLibrary
    ⊞ Condition
    ⊞ ControlActions
    ⊟ ControlAttachment
        Equals(Object) [bool]
        GetHashCode() [int]
        GetType() [Type]
        ToString() [string]
        ActivationState [AttachmentState]
        AttachmentObject [AttachmentObject]
    ⊞ ControlMacro
⊞ {} Shared
```

## Control Latch

Control latch modules provide the functionality associated with latching actions.

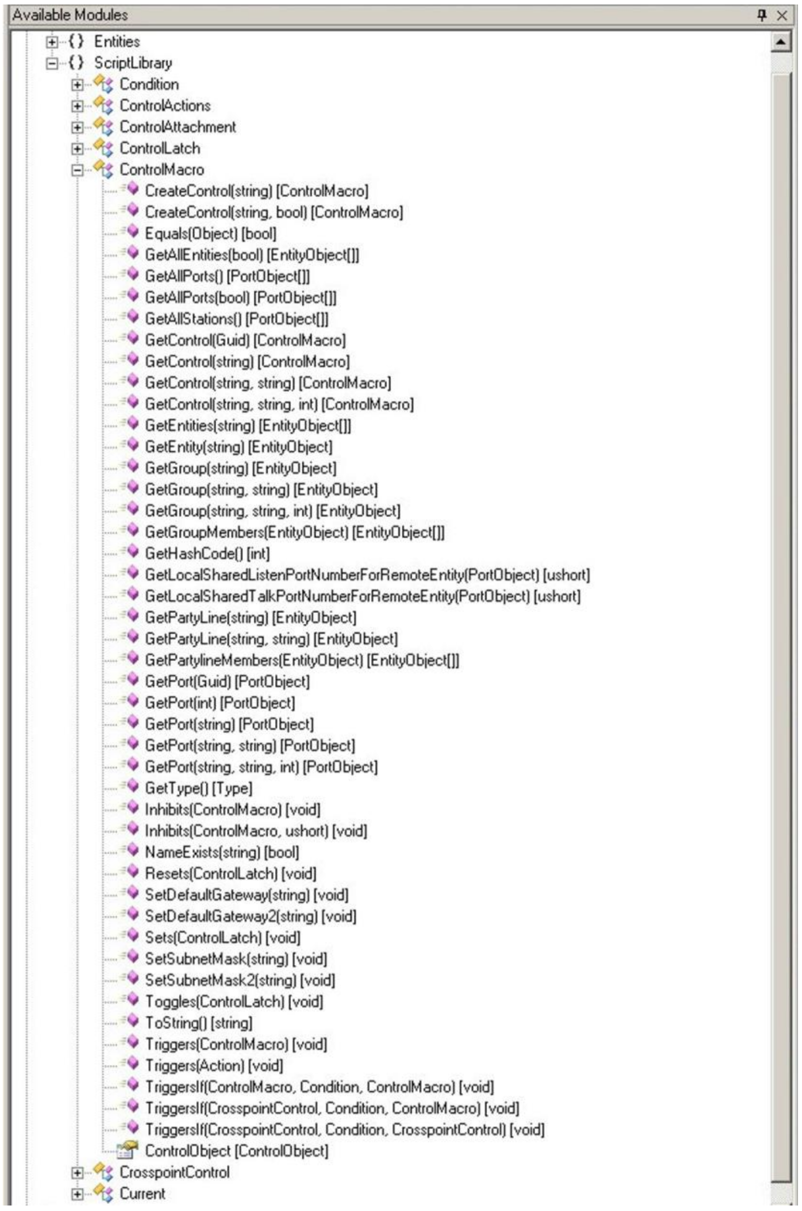


```
⊟ {} ClearCom
  ⊞ {} Entities
  ⊟ {} ScriptLibrary
    ⊞ Condition
    ⊞ ControlActions
    ⊞ ControlAttachment
    ⊟ ControlLatch
        CreateLatch() [ControlLatch]
        Equals(Object) [bool]
        GetHashCode() [int]
        GetType() [Type]
        ResetsWhenOff(ControlLatch) [void]
        ResetsWhenOn(ControlLatch) [void]
        SetsWhenOff(ControlLatch) [void]
        SetsWhenOn(ControlLatch) [void]
        TogglesWhenOff(ControlLatch) [void]
        TogglesWhenOn(ControlLatch) [void]
        ToString() [string]
        TriggersWhenOff(ControlMacro) [void]
        TriggersWhenOff(Action) [void]
        TriggersWhenOn(ControlMacro) [void]
        TriggersWhenOn(Action) [void]
        GetId [ushort]
    ⊞ ControlMacro
    ⊞ CrosspointControl
    ⊞ Current
⊞ {} Shared
```

**Figure A-18: Control Latch Actions List**

## Control Macro

Control macros act on system components to get or set the states or attributes of those components.

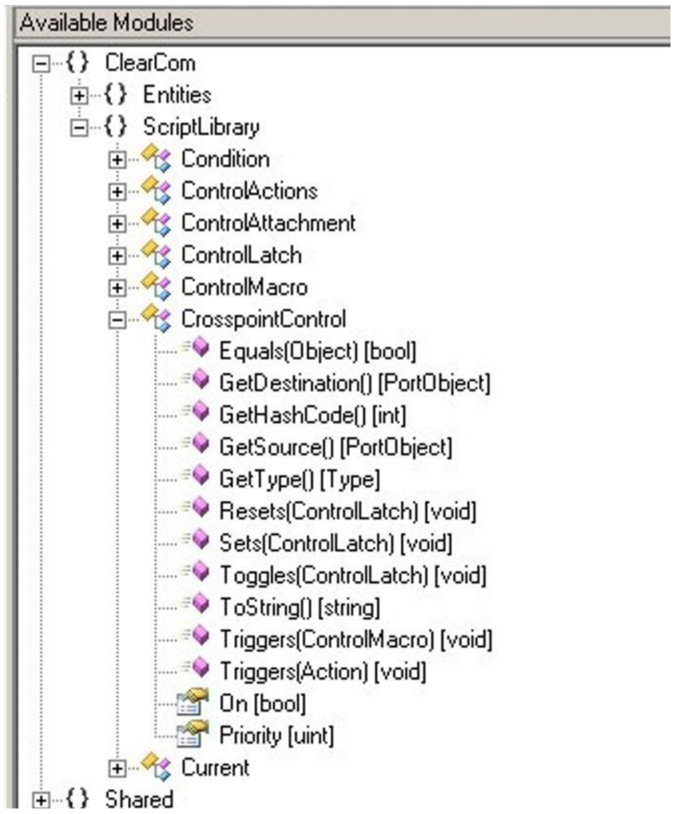


```
Available Modules
  {} Entities
  {} ScriptLibrary
     Condition
     ControlActions
     ControlAttachment
     ControlLatch
     ControlMacro
        CreateControl(string) [ControlMacro]
        CreateControl(string, bool) [ControlMacro]
        Equals(Object) [bool]
        GetAllEntities(bool) [EntityObject[]]
        GetAllPorts() [PortObject[]]
        GetAllPorts(bool) [PortObject[]]
        GetAllStations() [PortObject[]]
        GetControl(Guid) [ControlMacro]
        GetControl(string) [ControlMacro]
        GetControl(string, string) [ControlMacro]
        GetControl(string, string, int) [ControlMacro]
        GetEntities(string) [EntityObject[]]
        GetEntity(string) [EntityObject]
        GetGroup(string) [EntityObject]
        GetGroup(string, string) [EntityObject]
        GetGroup(string, string, int) [EntityObject]
        GetGroupMembers(EntityObject) [EntityObject[]]
        GetHashCode() [int]
        GetLocalSharedListenPortNumberForRemoteEntity(PortObject) [ushort]
        GetLocalSharedTalkPortNumberForRemoteEntity(PortObject) [ushort]
        GetPartyLine(string) [EntityObject]
        GetPartyLine(string, string) [EntityObject]
        GetPartylineMembers(EntityObject) [EntityObject[]]
        GetPort(Guid) [PortObject]
        GetPort(int) [PortObject]
        GetPort(string) [PortObject]
        GetPort(string, string) [PortObject]
        GetPort(string, string, int) [PortObject]
        GetType() [Type]
        Inhibits(ControlMacro) [void]
        Inhibits(ControlMacro, ushort) [void]
        NameExists(string) [bool]
        Resets(ControlLatch) [void]
        SetDefaultGateway(string) [void]
        SetDefaultGateway2(string) [void]
        Sets(ControlLatch) [void]
        SetSubnetMask(string) [void]
        SetSubnetMask2(string) [void]
        Toggles(ControlLatch) [void]
        ToString() [string]
        Triggers(ControlMacro) [void]
        Triggers(Action) [void]
        TriggersIf(ControlMacro, Condition, ControlMacro) [void]
        TriggersIf(CrosspointControl, Condition, ControlMacro) [void]
        TriggersIf(CrosspointControl, Condition, CrosspointControl) [void]
        ControlObject [ControlObject]
     CrosspointControl
     Current
```

**Figure A-19: Control Macro List**

## Crosspoint Control

Crosspoint controls act on system crosspoints to get or set the states of the crosspoints.

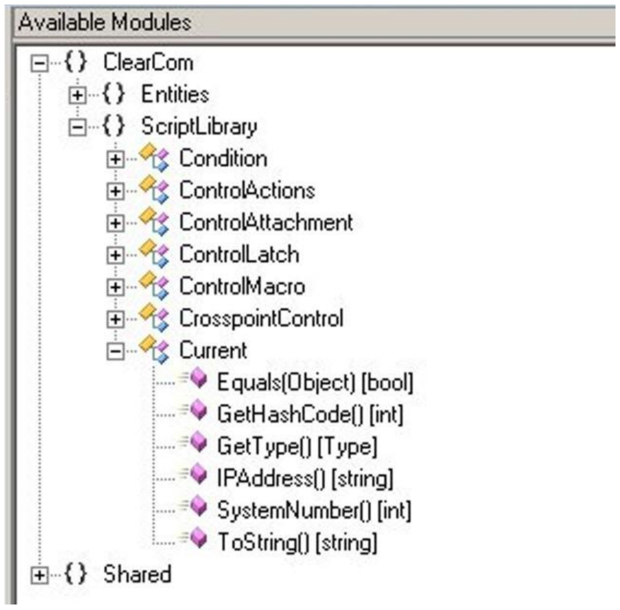


```
Available Modules
⊟ { } ClearCom
    ⊞ { } Entities
    ⊟ { } ScriptLibrary
        ⊞ Condition
        ⊞ ControlActions
        ⊞ ControlAttachment
        ⊞ ControlLatch
        ⊞ ControlMacro
        ⊟ CrosspointControl
            Equals(Object) [bool]
            GetDestination() [PortObject]
            GetHashCode() [int]
            GetSource() [PortObject]
            GetType() [Type]
            Resets(ControlLatch) [void]
            Sets(ControlLatch) [void]
            Toggles(ControlLatch) [void]
            ToString() [string]
            Triggers(ControlMacro) [void]
            Triggers(Action) [void]
            On [bool]
            Priority [uint]
        ⊞ Current
⊞ { } Shared
```

Figure A-20: Crosspoint Controls

## Current

Current provides facilities to obtain current system information.

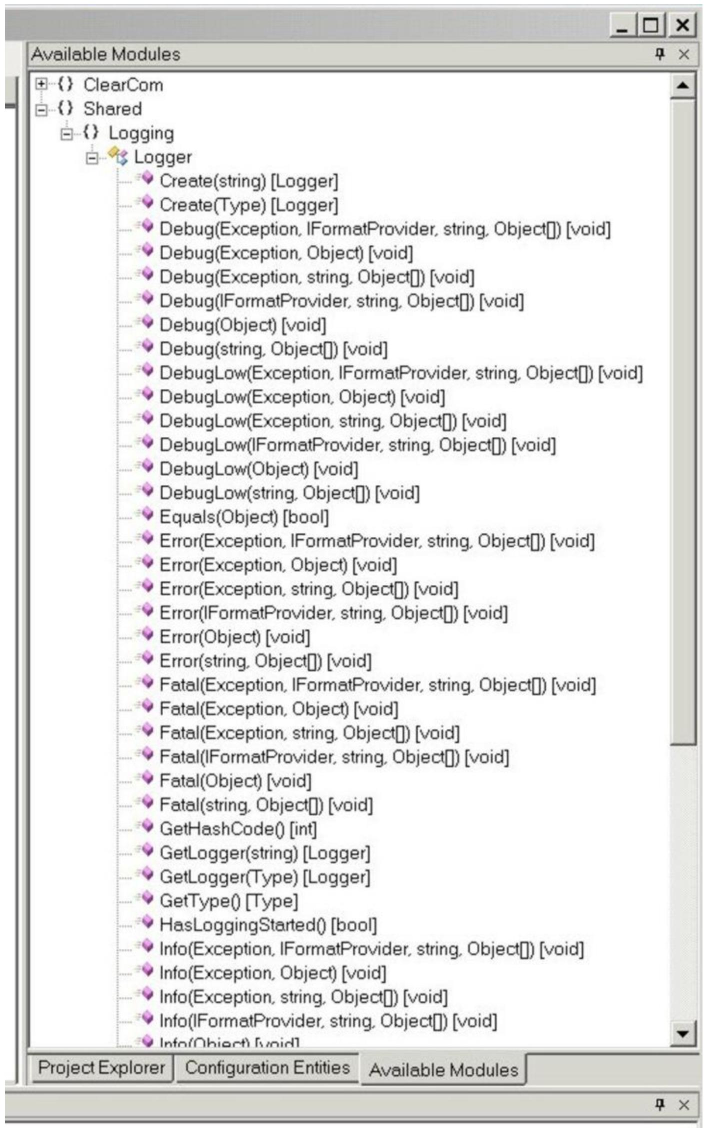


```
Available Modules
⊟ { } ClearCom
    ⊞ { } Entities
    ⊟ { } ScriptLibrary
        ⊞ Condition
        ⊞ ControlActions
        ⊞ ControlAttachment
        ⊞ ControlLatch
        ⊞ ControlMacro
        ⊞ CrosspointControl
        ⊟ Current
            Equals(Object) [bool]
            GetHashCode() [int]
            GetType() [Type]
            IPAddress() [string]
            SystemNumber() [int]
            ToString() [string]
⊞ { } Shared
```

Figure A-21: System Current

## 2.5.2 SHARED

The shared entry provides a library of objects for debugging control, error reporting, messages and logging from user control macros.

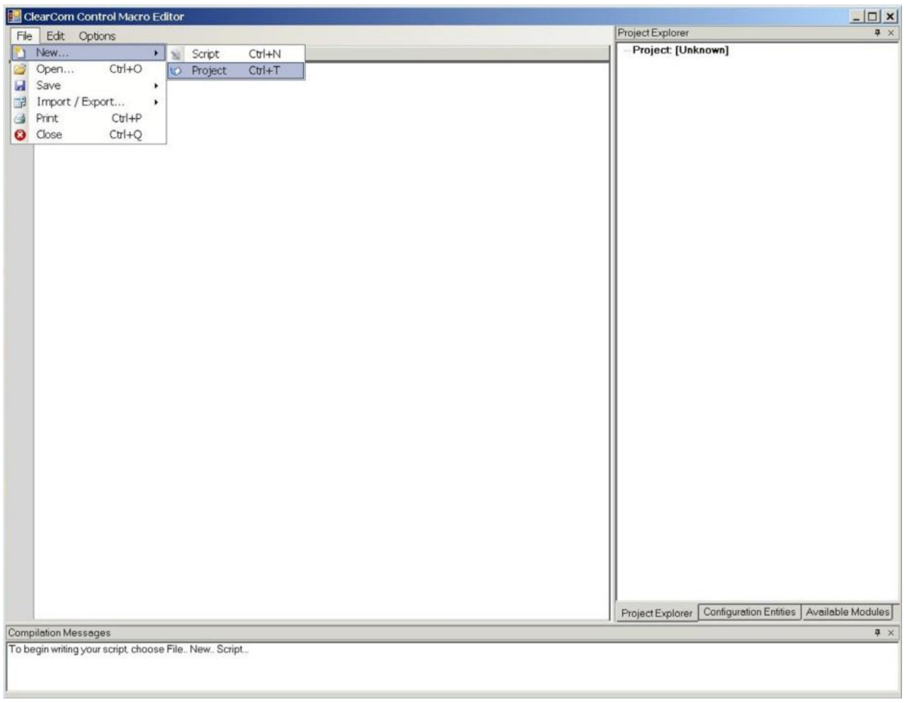


```
Available Modules
{} ClearCom
{} Shared
  {} Logging
    Logger
      Create(string) [Logger]
      Create(Type) [Logger]
      Debug(Exception, IFormatProvider, string, Object[]) [void]
      Debug(Exception, Object) [void]
      Debug(Exception, string, Object[]) [void]
      Debug(IFormatProvider, string, Object[]) [void]
      Debug(Object) [void]
      Debug(string, Object[]) [void]
      DebugLow(Exception, IFormatProvider, string, Object[]) [void]
      DebugLow(Exception, Object) [void]
      DebugLow(Exception, string, Object[]) [void]
      DebugLow(IFormatProvider, string, Object[]) [void]
      DebugLow(Object) [void]
      DebugLow(string, Object[]) [void]
      Equals(Object) [bool]
      Error(Exception, IFormatProvider, string, Object[]) [void]
      Error(Exception, Object) [void]
      Error(Exception, string, Object[]) [void]
      Error(IFormatProvider, string, Object[]) [void]
      Error(Object) [void]
      Error(string, Object[]) [void]
      Fatal(Exception, IFormatProvider, string, Object[]) [void]
      Fatal(Exception, Object) [void]
      Fatal(Exception, string, Object[]) [void]
      Fatal(IFormatProvider, string, Object[]) [void]
      Fatal(Object) [void]
      Fatal(string, Object[]) [void]
      GetHashCode() [int]
      GetLogger(string) [Logger]
      GetLogger(Type) [Logger]
      GetType() [Type]
      HasLoggingStarted() [bool]
      Info(Exception, IFormatProvider, string, Object[]) [void]
      Info(Exception, Object) [void]
      Info(Exception, string, Object[]) [void]
      Info(IFormatProvider, string, Object[]) [void]
      Info(Object) [void]
```

**Figure A-22: Shared Object List**

## 2.6 CREATING A NEW PROJECT

To start a new project click on 'File' and then 'New' to display the options to create a new control macro or project. Click on project to create a new project and the new project folder with the default name "Unknown" will be displayed in the object browser window.



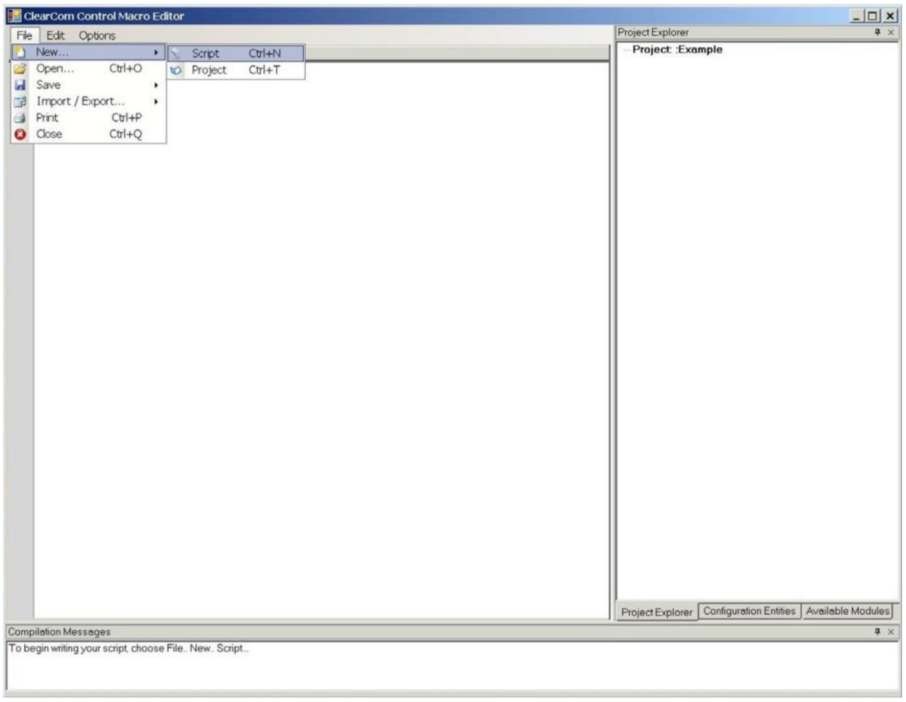
To begin writing your script, choose File.. New.. Script..

**Figure A-23: New Project Screen**

A project is a collection of different control macros, usually for a specific application such as a studio.

Double click on the ‘Project [Unknown]’ entry in the object browser to highlight it and right-click to display the command menu and select ‘Rename’ then type in the new project name. The new project can be saved by selecting ‘File’ and then ‘Save’ to save the project.

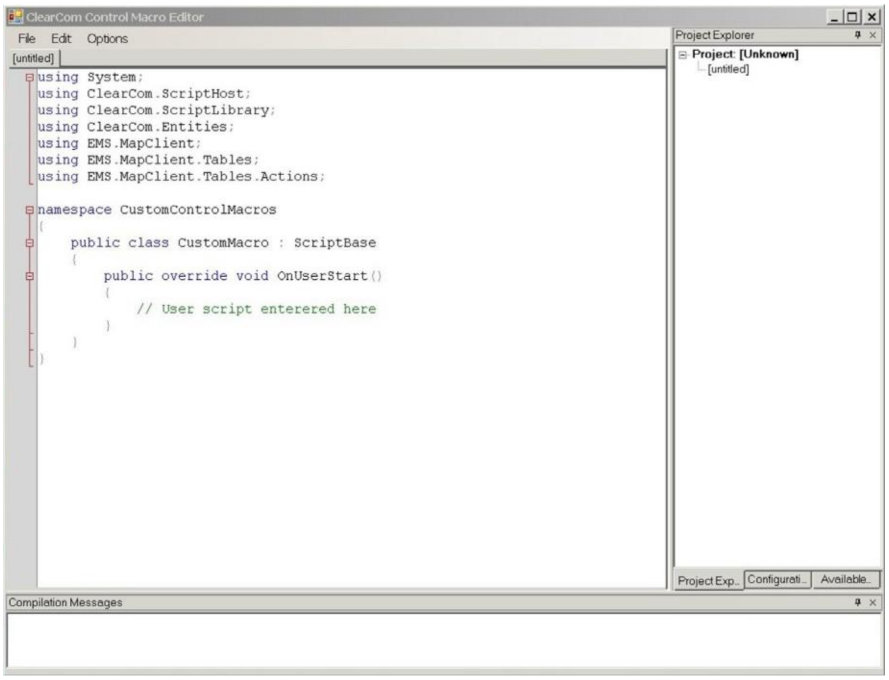
To start a new control macro click on ‘File’ to display the file menu, click on ‘New’ and then ‘Control Macro’ to initialize a new control macro.



To begin writing your script, choose File.. New.. Script..

**Figure A-24: Start New Control Macro**

After clicking on ‘Script’ the editor will automatically create the basic structure of the control macro with the required libraries set up at the start of the control macro. One the initial control macro has been created the user can start creating the application control macro under the comment ‘// User script entered here’.



```
using System;
using ClearCom.ScriptHost;
using ClearCom.ScriptLibrary;
using ClearCom.Entities;
using EMS.MapClient;
using EMS.MapClient.Tables;
using EMS.MapClient.Tables.Actions;

namespace CustomControlMacros
{
    public class CustomMacro : ScriptBase
    {
        public override void OnUserStart()
        {
            // User script enteredered here
        }
    }
}
```

**Figure A-25: Inital New Control Macro**

Once the control macro structure has been generated the user control macro is created by dragging and dropping items from the object browser into the control macro edit window to build up lines of the control macro.

For example, to create an instance of a port select the 'Configuration Entities' tab and open the 'PORT' item to display a list of ports. Right click on the required port to select it and then double click to automatically create the line of code that will create and instance of that object.

**Note:** Note: Enter some blank lines (keyboard Enter) after automatically generated '// user Script entered here' line

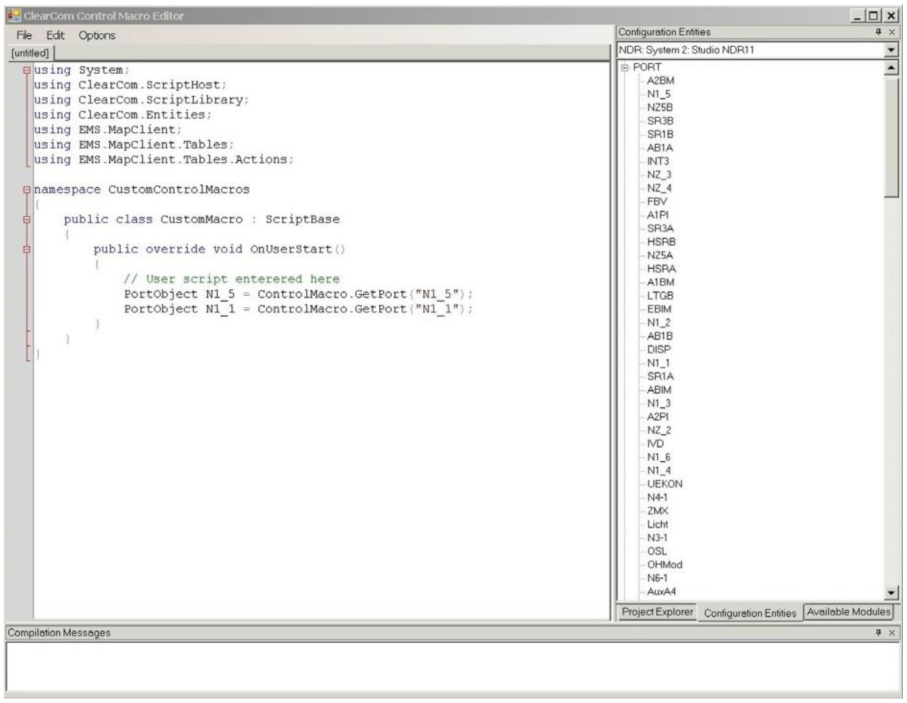
**Note:** Note: Make sure the cursor is placed on a line under the start of the user script marker before selecting a new control control macro line.

Certain types of macro actions may have variable or unpredictable effects on different types of hardware so where a macro may act on different types of hardware it should be checked on all the variants of the hardware.

Once such case is macros which cause LEDs on panels to flash. There are a variety of different panel types which may be present on a system and they may respond differently to commands to flash LEDs. For example a macro to cause LEDs to flash system wide will not work on ICS-2003 panels but will work on other panels. Macros which flash

LEDs at various frequencies may work on some panels but not on others. Generally a 1Hz flash is likely to work.

Macros may also reference keys on panels but it should be noted that the key numbering is different on different panels so any control macro will need to take account of this if there is more than one type of panel on a system. The key numbering on the various panel types is given in Appendix C.



```
using System;
using ClearCom.ScriptHost;
using ClearCom.ScriptLibrary;
using ClearCom.Entities;
using EMS.MapClient;
using EMS.MapClient.Tables;
using EMS.MapClient.Tables.Actions;

namespace CustomControlMacros
{
    public class CustomMacro : ScriptBase
    {
        public override void OnUserStart()
        {
            // User script enterered here
            PortObject N1_5 = ControlMacro.GetPort("N1_5");
            PortObject N1_1 = ControlMacro.GetPort("N1_1");
        }
    }
}
```

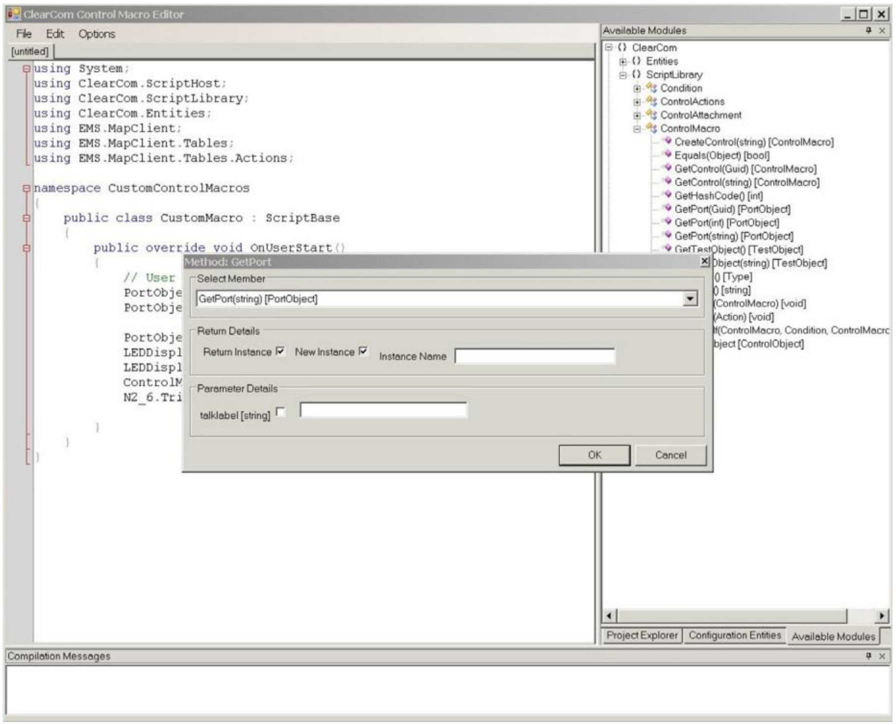
**Figure A-26: Control Macro with Port Commands**

In this way commands to create instances of configuration objects can be created. These instances can then to referenced by other commands to modify the source system configuration.

The user may also create control macros manually using a text editor such as Notepad but this is not normally recommended as the error checking facilities of the control macro editor will not be available.

When a control macro is dragged and dropped into the control macro editor window a configuration window is opened to request the parameters that are required for that control macro. Where there are a number of predetermined values for a parameter such as TRUE or FALSE a drop-down menu allows a parameter to be selected.

Alternatively a parameter name can be entered manually.



```
using System;
using ClearCom.ScriptHost;
using ClearCom.ScriptLibrary;
using ClearCom.Entities;
using EMS.MapClient;
using EMS.MapClient.Tables;
using EMS.MapClient.Tables.Actions;

namespace CustomControlMacros
{
    public class CustomMacro : ScriptBase
    {
        public override void OnUserStart()
        {
            // User
            PortObje
            PortObje

            PortObje
            LEDDispl
            LEDDispl
            ControlM
            N2_6.Tri
        }
    }
}
```

```
ClearCom
  Entities
  ScriptLibrary
    Condition
    ControlActions
    ControlAttachment
    ControlMacro
      CreateControl(string) [ControlMacro]
      Equals(Object) [bool]
      GetControl(Guid) [ControlMacro]
      GetControl(string) [ControlMacro]
      GetHashCode() [int]
      GetPort(Guid) [PortObject]
      GetPort(int) [PortObject]
      GetPort(string) [PortObject]
      GetTestObject() [TestObject]
      Object(string) [TestObject]
      () [Type]
      () [string]
      (ControlMacro) [void]
      (Action) [void]
      If(ControlMacro, Condition, ControlMacrc
      bject [ControlObject]
```

Select Member: GetPort(string) [PortObject]

Return Details: Return Instance ☑ New Instance ☑ Instance Name

Parameter Details: talklabel [string] ☐

**Figure A-27: Macro Parameter Entry Window**

When the parameters have been input clicking on ‘OK’ writes the new line into the control macro editor window at the current cursor position. Any errors in the command for example as a result of an incorrect parameter will be reported in the compilation messages window below the control macro editor window.

## 2.7 ELEMENTS OF A CONTROL MACRO

There are three basic steps to create a control function using the control macro facility. These are:

1) Set the objects the macros are to operate on. These may be ports or entities such as groups or conferences. For example, to create a port object that references a specific port select the ‘Configuration Entities’ tab in the object browser window and open the ‘Port’ item to display a list of ports in the current configuration. Double click on the required port to create the macro in the control macro editor window e.g.

PortObject var_myPortt = ControlMacro.GetPort(“MyPort”);

where ‘MyPort’ is the name of the port defined in EHX. Alternatively port objects can be created by selecting the ‘Available Modules’ tab in the object browser and opening the Scriptlibrary.ControlMacro menu and selecting the ‘Getport (string)’ macro. Ports may be selected by port name (string parameter), port number (integer parameter) or by global identifier (Guid).

2) Create an action to perform. Select the 'Available Modules' tab in the object browser window and open the Scriptlibrary ControlActions menu. Actions which use the objects previously created can be dragged and dropped into the control macro. For example the action to activate an LED can be created using a port object created in step 1.

```
LEDDisplayAction MyLED = ControlAction.ActivateLED(MyPortObject,1);
```

will create an action 'MyLED' to activate the LED on key 1 on a panel attached to port 'MyPort'.

3) Create a control object which will be used to trigger the action set up in step 2. For example a control action could be created using a general purpose I/O port by selecting the 'Configuration Entities' tab and opening the 'GPSF' item to display a list of GPSF items. Double click on the required item to create the control macro in the control macro editor window e.g.

```
ControlMacro MyControl = ControlMacro.GetControl("MyGPSF");
```

4) Trigger the action. To do this a control must be created which connects an event on the system with the action that has been created. For example, a control can be created for another port e.g.

```
MyControl.Triggers(MyLED);
```

so that an event on the GPSF 'MyGPSF' will trigger the LED on key 1 of the panel attached to 'MyPort'.

## 2.8 MACRO REFERENCE

The objects from the Available Modules are described in this section. These macros are used to construct control macros using the control macro editor. The meanings of the parameters used by the macros are:

- () - required parameter(s)
- [] - type of argument returned
- object - the name of the object being tested, normally an object created by a control macro such as 'GetPort'
- bool - boolean operator, set to True or False
- int - integer value in the range 0 - 32767
- string - alphanumeric string parameter
- Guid - an EHX internal global identifier. Every entity has a unique internal identifier and while these may be used as input parameters for some control macros they are not generally used.

## 2.8.1 ATTACHMENTOBJECT MACROS

These macros are accessed by expanding the 'Clearcom' > 'Entities' > 'AttachmentObject' entry in the Available Modules menu.

| Macro | Description |
|---|---|
| Equals (object) [bool] | Tests the equivalence of two objects and returns True or False. e.g. **bool <result> = <object1>.equals<(object2)>;** |
| GetHashCode () [int] | Returns the hash code of an object previously created by a control as an integer. e.g. **int <result> = <object>.GetHashCode();** |
| GetType () [Type] | Returns the type of an object previously created by a control macro. e.g. **Type <result> = <object>.GetType();** |
| ToString () [string] | Returns the string value of an object previously created by a control macro. e.g. **string <result> = <object>.ToString();** |
| ActivateWithListen [bool] | Either returns the listen status of an object created by a control macro as a boolean True or False or sets the lis- ten status of an object to a boolean True or False e.g. **AttachmentObject <result> = <object>.ActivateWith- Listen;** or **<object>.ActivateWithListen = <listen state>;** |

| Macro | Description |
|---|---|
| IsStnRelay [bool] | Either returns whether the status of an object created by a control macro is a station relay as a boolean True or False or sets the status of an object as a station relay to a boolean True or False e.g.<br>**AttachmentObject <result> = <object>.IsStnRelay;**<br>or<br>**<object>.IsStnRelay = <boolean>;** |
| ModuleNumber [int] | Either returns the module number of an object created by a control macro as an integer value or sets the mod- ule number of an object to an integer value e.g.<br>**AttachmentObject <integer> = <object>.Module- Number;**<br>or<br>**<object>.ModuleNumber = <integer>;** |
| OutputNumber [int] | Either returns the output number of an object created by a control macro as an integer value or sets the output number of an object to an integer value e.g.<br>**AttachmentObject <integer> = <object>.OutputNum- ber;**<br>or<br>**<object>.OutputNumber = <integer>;** |
| RelayInformation [string] | Either returns the relay information of an object created by a control macro as a string or sets the relay informa- tion of an object to a string e.g.<br>**AttachmentObject <string> = <object>.RelayInfor- mation;**<br>or<br>**<object>.RelayInformation = <string>;** |

| Macro | Description |
|---|---|
| ReverseListen [bool] | Either returns the reverse listen status of an object cre- ated by a control macro as a boolean True or False or sets the reverse listen status of an object to a boolean True or False e.g. **AttachmentObject <boolean> = <object>.ReverseListen** or **<object>.ReverseListen = <boolean>** |
| RouteDestID [Guid] | Either returns the route destination ID of an object cre- ated by a control macro as type Guid or sets the route destination ID of an object to a Guid e.g. **AttachmentObject <destID> = <object>.RouteDestID** or **<object>.RouteDestID = <destID>** |
| RouteSourceID [Guid] | Either returns the route source ID of an object created by a control macro as type Guid or sets the route source ID of an object to a Guid e.g. **AttachmentObject <sourceID> = <object>.Route- SourceID** or **<object>.RouteSourceID = <sourceID>** |

## 2.9 CONTROL OBJECT MACROS

These macros are accessed by expanding the 'Clearcom' > 'Entities' > 'ControlObject' entry in the Available Modules menu.

| Macro | Description |
|---|---|
| Dispose () [void] | Disposes of an object created by a control macro e.g. **<object>.Dispose();** |

| Macro | Description |
|---|---|
| Equals (Object) [bool] | Tests the equivalence of two objects and returns True or False. e.g:<br>**bool <result> = <object1>.equals<(object2)>;** |
| GetGPSF () [GPSF] | Gets the GPSF e.g.<br>**GPSF <result> = <object>.GetGPSF();** |
| GetGPSF (TalkType) [GPSF] | Gets the talk type for GPSF e.g.<br>**GPSF <result> = <object>.GetGPSF(<talk type>);** |
| GetHashCode () [int] | Returns the hash code of an object previously created by a control as an integer. e.g:<br>**int <result> = <object>.GetHashCode();** |
| GetID (TalkType) [Guid] | Returns the Guid of the talk/listen status of an object created by a control macro e.g.<br>**Guid <return> = <object>.GetID(<talk/listen type>);** |
| GetOwnerSystemGPSF(Talk-Type) [GPSF] | Returns the GPSF of an object specified by TalkType e.g.<br>**GPSF <return> = <object>.GetOwnerSystemG-PSF(TalkType.<talk/listen type>);** |
| GetOwnerSystemRackOff-set(TalkType) [ushort] | Returns the rack number of an object specified by Talk- Type e.g.<br>**ushort <return> = <object>.GetOwnerSystemRack-Offset(TalkType.<talk/listen type>);** |
| GetRackOffset () [ushort] | Returns an offset value as an unsigned short for the object previously created by a control macro e.g.<br>**ushort <value> = <object>.GetRackOffset();** |
| GetRackOffset (TalkType) [ush-ort] | Returns an offset value as an unsigned short for the object previously created by a control macro where the type of route is specified ie Talk and/or Listen e.g.<br>**ushort <value> = <object>.GetRackOffset(<type>);** |

| Macro | Description |
|---|---|
| SetGPSF (TalkType, GPSF) [void] | Sets the talktype GPSF e.g. **GPSF <result> = <object>.SetGPSF(<talk type>,<GPSF>);** |
| SetID (Guid, TalkType) [void] | Sets the ID of the talk/listen status of an object created by a control macro e.g. **Guid <return> = <object>.GetID(<talk/listen type>);** |
| SetOwnerSystemGPSF(Talk-Type,GPSF) [void] | Sets the GPSF of the object specified by TalkType e.g. **<object>.SetOwnerSystemGPSF(TalkType.<talk/lis- ten type>, GPSF);** |
| TalkTypeIsCreated(TalkType) [bool] | Creates an object TalkType and returns the result as a boolean e.g. **bool <result> = <object>.TalkTypeIsCreated(Talk-Type.<talk/listen type>);** |
| ToString () [string] | Returns the string value of an object previously created by a control macro. e.g. **string <result> = <object>.ToString();** |
| ConfigurationID [Guid] | Sets or returns the configuration ID of an object created by a control macro as a control object e.g. **ControlObject <ID object> = <object>.Configura- tionID;** or **<object>.ConfigurationID = <ID object>;** |
| EntityID [Guid] | Sets or returns the entity ID of an object created by a control macro as a Guid e.g. **ControlObject <Guid> = <object>.EntityID;** or **<object>.EntityID = <Guid>;** |
| EntityType [dest_type_e] | Returns the entity type of an object created by a control macro as a Guid e.g. **ControlObject <object> = <object>.EntityType;** |

| Macro | Description |
|---|---|
| IsCreated [bool] | Returns a boolean indicating whether an object created by a control macro has been created e.g.<br>**ControlObject <bool> = <object>.IsCreated;** |
| LatchDisable [bool] | Sets or returns the latch disable status of an object cre- ated by a control macro e.g.<br>**ControlObject <bool> = <object>.LatchDisable;**<br>or<br>**<object>.LatchDisable = <bool>;** |
| ListenAlias [string] | Sets or returns the listen alias of an object created by a control macro e.g.<br>**ControlObject <string> = <object>.ListenAlias;**<br>or<br>**<object>.ListenAlias = <string>;** |
| ListenLabel [string] | Sets or returns the listen label of an object created by a control macro e.g.<br>**ControlObject <string> = <object>.ListenLabel;**<br>or<br>**<object>.ListenLabel = <string>;** |
| TalkAlias [string] | Sets or returns the talk alias of an object created by a control macro e.g.<br>**ControlObject <string> = <object>.TalkAlias;**<br>or<br>**<object>.TalkAlias = <string>;** |
| TalkLabel [string] | Sets or returns the talk label of an object created by a control macro e.g.<br>**ControlObject <string> = <object>.TalkLabel;**<br>or<br>**<object>.TalkLabel = <string>;** |
| GetType () [Type] | Returns the type of an object previously created by a control macro. e.g.<br>**Type <result> = <object>.GetType();** |

## 2.9.1 PORT OBJECT MACROS

These macros are accessed by expanding the 'Clearcom' > 'Entities' > 'PortObject' entry in the Available Modules menu.

| Macro | Description |
|---|---|
| Equals (Object) [bool] | Tests the equivalence of two objects and returns True or False. e.g:<br>**bool <result> = <object1>.equals<(object2)>** |
| GetHashCode () [int] | Returns the hash code of an object previously created by a control as an integer. e.g:<br>**int <result> = <object>.GetHashCode()** |
| GetType () [Type] | Returns the type of an object previously created by a control macro. e.g.<br>**Type <result> = <object>.GetType()** |
| ToString () [string] | Returns the string value of an object previously created by a control macro. e.g.<br>**string <result> = <object>.ToString();** |
| CC_ADV_TYPE [Destination Type] | Gets or sets the port type according to the parameter details selected from a menu e.g.<br>**<port object>.CC_ADV_TYPE = <Destination Type>.CC_ADV_PORT;** |
| CombinedLabel[string] | Returns the port Talk/Listen label specified by the string e.g.<br>**PortObject <resultstring> = <port object>.Combin- edLabel;** |
| ConfigurationID [Guid] | Sets or returns the configuration ID of an object created by a control macro as a control object e.g.<br>**ControlObject <ID object> = <object>.Configura- tionID;**<br>or<br>**<object>.ConfigurationID = <ID object>;** |

| Macro | Description |
|---|---|
| Description [string] | Sets or returns the description of a port as a string e.g.<br>**<port object>.Description = <description string>;**<br>or<br>**PortObject <string> = <port** |
| EnableTalkAssign [bool] | Sets or returns the permission to assign a port as Listen using a boolean e.g.<br>**<port object>.EnableTalkAssign = True;**<br>or<br>**PortObject <boolean> = <port object>.Enable-TalkAssign;** |
| EntityID [Guid] | Sets or returns the entity ID of an object created by a control macro as a Guid e.g.<br>**ControlObject <Guid> = <object>.EntityID;**<br>or<br>**<object>.EntityID = <Guid>;** |
| EntityType [dest_type_e] | Returns the entity type of an object created by a control macro as a Guid e.g.<br>**ControlObject <object> = <object>.EntityType;** |
| GlobalIfb [bool] | Gets or sets the global IFB (Interruptable foldback) on a port e.g.<br>**<port object>.GlobalIfb = True;**<br>or<br>**PortObject <boolean result> = <port object>.Glo- balIfb;** |
| GlobalIso [bool] | Gets or sets the global ISO on a port e.g.<br>**<port object>.GlobalIso = <boolean>;**<br>or<br>**PortObject <boolean result> = <port object>.Glo- balIso;** |

| Macro | Description |
|---|---|
| LatchDisable [bool] | Sets or returns the latch disable status of an object cre- ated by a control macro e.g.<br>**ControlObject <bool> = <object>.LatchDisable;**<br>or<br>**<object>.LatchDisable = <bool>;** |
| ListenAlias [string] | Sets or returns the listen alias of an object created by a control macro e.g.<br>**ControlObject <string> = <object>.ListenAlias;**<br>or<br>**<object>.ListenAlias = <string>;** |
| ListenLabel [string] | Sets or returns the listen label of an object created by a control macro e.g.<br>**ControlObject <string> = <object>.ListenLabel;**<br>or<br>**<object>.ListenLabel = <string>;** |
| PortNumber [ushort] | Sets or returns the port number of an object created by a control macro as an unsigned short e.g.<br>**ushort <portno> = <object>.PortNumber;**<br>or<br>**<object>.PortNumber = <portno>;** |
| PortSubType [EntityType] | Sets or returns the port subtype of an object created by a control macro as a port subtype entity e.g.<br>**PortObject <portsubtype> = <object>.PortSubType;**<br>or<br>**<object>.PortSubType = <portsubtype>;** |
| PortType [BasicType] | Sets or returns the port type of an object created by a control macro as a port type entity e.g.<br>**PortObject <type> = <object>.PortType;**<br>or<br>**<object>.PortType = BasicType.<type>;** |

| Macro | Description |
|---|---|
| PreventReplySignalisation [bool] | Sets the status of the prevent reply signalization setting for the port using the boolean e.g.<br>**<port object>.PreventReplySignalization = True;** |
| PreventTally [bool] | Sets or returns the status of the prevent tally setting for the port using the boolean e.g.<br>**<port object>.PreventTally = False;**<br>or<br>**PortObject <result> = <port object>.PreventTally;** |
| ProtectPortAssignment [bool] | Sets or returns the status of the port protection setting for the port using the boolean e.g.<br>**<port object>.ProtectPortAssignment = True;**<br>or<br>**PortObject protectPortAssignment = <port object>.ProtectPortAssignment;** |
| SecondaryAction [Guid] | Sets or returns the secondary action of an object cre- ated by a control macro e.g.<br>**Guid <return> = <object>.SecondaryAction;**<br>or<br>**<object>.SecondaryAction = <Guid>;** |
| SplitLabel [bool] | Sets or returns the split label status of a port using the boolean e.g.<br>**<port object>.SplitLabel = True;**<br>or<br>**PortObject <relult boolean> = <port object>.SplitLa- bel;** |
| StackedKey [bool] | Sets or returns a boolean indicating whether a key is a stacked key e.g.<br>**bool <return> = <object>.StackedKey;**<br>or<br>**<object>.StackedKey = <bool>;** |

| Macro | Description |
|---|---|
| TalkAlias [string] | Sets or returns the talk alias of an object created by a control macro e.g.<br>**ControlObject <string> = <object>.TalkAlias;**<br>or<br>**<object>.TalkAlias = <string>;** |
| TalkLabel [string] | Sets or returns the talk label of an object created by a control macro e.g.<br>**ControlObject <string> = <object>.TalkLabel;**<br>or<br>**<object>.TalkLabel = <string>;** |
| VoxAction [Guid] | Sets or returns the vox action of an object created by a control macro e.g.<br>**Guid <Guid> = <object>.VoxAction;**<br>or<br>**<object>.VoxAction = <Guid>;** |
| EnableListenAssign [bool] | Sets or returns the permission to assign a port as Listen using a boolean e.g.<br>**<port object>.EnableListenAssign = True;**<br>or<br>**PortObject <boolean> = <port object>.EnableLis- tenAssign;** |

## 2.9.2 CONDITION MACROS

These macros are accessed by expanding the 'Clearcom' > 'ScriptLibrary' > 'Condition' entry in the Available Modules menu.

| Macro | Description |
|---|---|
| CompareTo (Object) [int] | Returns an integer value from the comparison of two objects e.g.<br>**int <result> = <object1>.CompareTo(<object2>);** |

| Macro | Description |
|---|---|
| Equals (Object) [bool] | Tests the equivalence of two objects and returns True or False. e.g: **bool <result> = <object1>.equals<(object2)>;** |
| GetHashCode () [int] | Returns the hash code of an object previously created by a control as an integer. e.g: **int <result> = <object>.GetHashCode();** |
| GetType () [Type] | Returns the type of an object previously created by a control macro. e.g. **Type <result> = <object>.GetType();** |
| GetTypeCode () [TypeCode] | Returns the type code of an object previously created by a control macro. e.g. **TypeCode <result> = <object>.GetTypeCode();** |
| ToString () [string] | Returns the string value of an object previously created by a control macro. e.g. **string <result> = <object>.ToString();** |
| ToString (IFormatProvider) [string] | Returns the string value of an object previously created by a control macro formatted by a format specifier e.g. **string <result> = <object>.ToString(<format>);** |
| ToString (string) [string] | Returns the string value of an object previously created by a control macro formatted by a format provded as a parameter e.g. **string <result> = <object>.ToString(<format>);** |
| ToString (string, IFormatPro- vider) [string] | Returns the string value of an object previously created by a control macro formatted by a format provded as two parameters e.g. **string <result> = <object>.ToString(<format>,<for- mat>);** |
| AND [Condition] | Specifies a condition to tested between two objects and returns a boolean TRUE or False e.g. **<object1>,Condition.AND,<object2>;** |
| value [int] | Returns the value of a condition eg. **Condition <result> = <condition>.value ;** |
| OR [Condition] | Specifies a condition to tested between two objects and returns a boolean TRUE or FALSE eg. **<object1>,Condition.OR,<object2>;** |

## 2.9.3 CONTROL ACTIONS MACRO

Control action macros act on system configuration objects to change the state of the object. The format of a control action macro command is:

ControlActions.<Macroname><parameters>;

The 'ControlActionMacro' command is used to change the state of a system configuration object

For example, the command:

Action fireLed1 = ControlActions.ActivateLED(<parameters>);

will create an action 'fireLED1' that changes the state of the system object LED1 in accordance with the parameters supplied.

These macros are accessed by expanding the 'Clearcom' > 'ScriptLibrary' > 'ControlActions' entry in the Available Modules menu.

| Macro | Description |
|---|---|
| ActivateLED(EntityObject, LedRate, LedIndication) [Action] | Returns a control object to set the flash rate and color for a specified LED e.g.<br>**Action = ControlActions.ActivateLED(<EntityOb- ject>, Ledrate, Off, LedIndications.Red);** |
| ActivateLED (EntityObject[], LedRate, LedIndication) [Action] | Returns a control object to set the flash rate and color for specified LED e.g.<br>**Action = ControlActions.ActivateLED(<EnityOb- ject>, LedRate.Off, LedIndictions.Red);** |
| ActivateLED (PortObject, ush- ort) [LEDDisplayAction] | Returns a control object to activate a LED on a specified port and key number e.g.<br>**LEDDisplayAction <result> = ControlActions.Acti- vateLED(<port object>,<key number>);** |
| ActivateLed (PortObject, ushort, ushort, ushort) [LEDDisplayAc- tion] | Returns a control object to activate a LED on a specified port, key number, key region and key page e.g.<br>**LEDDisplayAction <result> = ControlActions.Acti- vateLED(<port object>,<key number>, key region>,<key page>);** |

| Macro | Description |
|---|---|
| ActivateLED (PortObject, ush- ort, ushort, ushort, LedRate, LedIndication) {LEDDisplayAc- tion] | Returns a control object to activate a LED on a specified port, key number, key region, key page, LED rate and LED colour e.g. **LEDDisplayAction <result> = ControlActions.Acti- vateLED(<port object>,<key number>, key region>,<key page>, <LED rate>,<LED colour>);** The parameters <LED rate> and <LED colour> may be selected from a drop-down menu or specified as a number. |
| ActivateLED (PortObject, ush- ort, ushort, ushort, bool, LedRate, LedIndication) [LED- DisplayAction] | Returns a control object to activate a LED on a specified port, key number, key region, key page, LED rate and LED colour e.g. **DisplayAction <action name> = ControlActions.Acti- vateLED(<port name>, <key number>, <Key Region>, <Key Page>, LedRate.Off, LedIndica- tion.Green);** The parameters <LED rate> and <LED colour> may be selected from a drop-down menu or specified as a number. |
| CallSignalAction () [CallSigna- lAction] | Returns an object that can be used to call action func- tions e.g. **CallSignalAction <object> = ControlActions.CallSig- nalAction ();** |
| Control (ControlMacro) [Action] | Returns the result of a control action e.g. **Action <result> = ControlActions.Control(<action>);** |
| Control (ControlMacro, Bits) [Action] | Returns the result of a control action e.g. **Action <result> = ControlActions.Control(<action>, <control bits>);** |
| CrossPointAction () [Cros- sPointAction] | Returns a crosspoint action e.g. **Action <result> = ControlActions.CrosspointAc- tion();** |

| Macro | Description |
|---|---|
| ActivateLED (PortObject, Entity-Object, LedRate, LedIndication) [Action] | Returns a control object to set the flash rate and color for a specified LED on a specified port e.g. **Action = ControlActions.ActivateLED(<port object>, <Entity Object>, LedRate.Off, LedIndication.Red);** |
| CutLoudspeaker (PortObject) [ControlMacro] | Cuts the loudspeaker on the specified port e.g. **ControlMacro = ControlActions.CutLoud-Speaker(<port object>);** |
| DCCAction (ushort, ushort, int) [Action] | Returns a Digital Control Card (DCC) action e.g. **Action <result> = ControlActions.DCCAction();** |
| Equals (Object) [bool] | Tests the equivalence of two objects and returns True or False. e.g. **bool <result> = <object1>.equals<(object2)>;** |
| FrameRelay (ushort) [Digital-ControlCardAction] | Returns a relay action object for a specific relay on a digital control card e.g. **DigitaControlCardAction <action> = ControlAc-tions.FrameRelay(<relay number>);** |
| GetHashCode () [int] | Returns the hash code of an object previously created by a control as an integer. e.g: **int <result> = <object>.GetHashCode();** |
| GetLogic (bool, bool, bool, bool, bool, bool) [Bits] | Returns a bit pattern corresponding to the boolean vari- ables e.g. **Bits <bit pattern> = ControlActions.GetLogic (<bool>,<bool>,<bool>,<bool>,<bool>,<bool>);** |
| GetType () [Type] | Returns the type of an object previously created by a control macro. e.g. **Type <result> = <object>.GetType();** |
| HeadsetSelect (portObject) [ControlMacro] | Selects the headset on the specified port e.g. **ControlMacro = ControlActions.HeadsetSelect)(Por- tObject);** |

| Macro | Description |
|---|---|
| IsolateRoute (PortObject, Port-Object) [IsolateAction] | Isolates a route between the two specified ports e.g. **IsolateAction = ControlActions.IsolateRoute(PortO- bject, PortObject);** |
| LatchResetAction (Control-Latch) [Action] | Resets the specified latch to the state specified in Action e.g. **Action = ControlActions.LatchResetAction(Control-Latch);** |
| LatchSetAction (ControlLatch] [Action] | Sets the specified latch to the state specified in Action e.g. **Action = ControlActions.LatchSetupAction(Control-Latch);** |
| LocalAction (PortObject, Loca-lAction) [LocalAction] | Returns a control for a local action on the specified port e.g. **LocalAction = ControlActions.LocalAction(PortOb- ject, LocalAction.Null);** |
| MicMute (PortObject) [Control-Macro] | Mutes the microphone on the specified port e.g. **ControlMacro = ControlActions.MicMute(PortOb-ject);** |
| NewDCCAction (ushort, ushort, int, bool) [Action] | Returns a Digital Control Card (DCC) action e.g. **Action <result> = ControlActions.NewDCCAc-tion(<card>,<pin>,<remote system>,<station relay>);** |
| RouteOff (PortObject, PortOb-ject) [Action] | Returns a new action for disabling a route between two ports e.g. **Action <result> = ControlAction.RouteOff (<source port object>,<destination port object>);** |
| RouteOff (ushort, ushort, ushort) [Action] | Returns a new action for disabling a route between two ports e.g. **Action <result> = ControlAction.RouteOff (<source portnumber >,<destination port number>,<source system number>);** |
| RouteOffPartyLine (PortObject, EntityObject) [Action] | Returns a new action for disabling a route between two party lines (conferences) e.g. **Action = ControlActions.RouteOffPartyLine(PortOb- ject, EntityObject);** |

| Macro | Description |
|---|---|
| RouteOn (PortObject, PortObject) [Action] | Returns a new action for enabling a route between two ports e.g. **Action <result> = ControlAction.RouteOn (<source port object>,<destination port object>);** |
| RouteOn (PortObject, ushort) [Action] | Returns a new action for enabling a specified route e.g. **Action = ControlActions.RouteOn(PortObject, <group number>);** |
| RouteOn (ushort, ushort, ushort) [Action] | Returns a new action for enabling a route between two ports e.g. **Action <result> = ControlAction.RouteOn(<source portnumber >,<destination port number>,<source system number>);** |
| LatchToggleAction (ControlLatch) [Action] | Toggles the specified latch to the state specified in Action e.g. **Action = ControlActions.LatchToggleAction(ControlLatch);** |

| Macro | Description |
|---|---|
| RouteToGroup (ushort, ushort, bool, bool, ushort) [Action] | Returns an action for creating a route between two groups e.g. **Action = ControlActions.RouteToGroup(<source port no>, <group offset number>, <talk or listen boolean>, <permanent boolean>, <remote system number>);** |
| RouteToGroupAction () [RouteToGroupAction] | Returns an action for creating a route to a group e.g. **RouteToGroupAction <result> = ControlActions.RouteTo GroupAction();** |
| RouteToGroupOn (ushort, ushort) [Action] | Returns an action for enabling a route between two groups e.g. **Action = ControlActions.RouteToGroupOn(<source port no>, <group number>);** |
| RouteToIfbOn (PortObject, PortObject) [Action] | Returns an action for enabling a route to an IFB e.g. **Action = ControlActions.RouteToIfbOn(<source port object>, <destination port object>);** |

| | |
|---|---|
| RouteToPartyLine (portObject, EntityObject) [Action] | Returns an action for enabling a route to a party line e.g. **Action = ControlActions.RouteToPartyLine(<source port object>, <party line object>);** |
| RouteToPartyLine (ushort, ushort, bool, bool, ushort) [Action] | Returns an action for enabling a route between two party lines e.g. **Action = ControlActions.RouteToPartyLine(<source port number>, <party line number>, <talk or listen boolean>, <permanent boolean>, remote system number>);** |
| RouteToPartyLineAction () [RouteToPartyLineAction] | Returns an action for creating a route to a party line e.g. **RouteToPartyLineAction <result> = ControlAc- tions.RouteTo PartyLineAction();** |
| SourceToIfbOn (PortObject, PortObject) [Action] | Creates an action to enable a crosspoint linking the source and destination ports for an IFB. **PortObject s = ControlMacro.GetPort(sourceport); PortObject d = ControlMacro GetPort(destport); Action action; action = ControlActions.SourceToIfbOn(s, d);** |
| SpeedDialAction (ushort, ushort) [Action] | Returns an action for creating a speed dial action with a speed dial ID and a port number e.g. **Action <result> = ControlAction.SpeedDialAction (<speed dial ID>,<port number>);** |
| ToString () [string] | Returns the string value of an object previously created by a control macro. e.g. **string <result> = <object>.ToString();** |
| SpeedDial (PortObject, string) [SpeedDialAction] | Returns an action for creating a speed dial with a port object and a telephone number e.g. **SpeedDialAction <result> = ControlAction.SpeedDi- alAction (<port object>,<telephone number>);** |

## 2.9.4 CONTROL ATTACHMENT MACROS

These macros are accessed by expanding the ‘Clearcom’ > ‘ScriptLibrary’ > ‘ControlAttachment’ entry in the Available Modules menu.

| Macro | Description |
|---|---|
| Equals (object) [bool] | Tests the equivalence of two objects and returns True or False. e.g. **bool <result> = <object1>.equals<(object2)>;** |
| GetHashCode () [int] | Returns the hash code of an object previously created by a control as an integer. e.g. **int <result> = <object>.GetHashCode();** |
| GetType () [Type] | Returns the type of an object previously created by a control macro. e.g. **Type <result> = <object>.GetType();** |
| ToString () [string] | Returns the string value of an object previously created by a control macro. e.g. **string <result> = <object>.ToString();** |
| ActivationState [Attachment-State] | Gets or sets the listen activation state of an object e.g. **<object>.ActivateWithListen = <boolean>;** |
| AttachmentObject [Attachment-Object] | Gets or sets the talk activation state of an object e.g. **<object>.ActivateWithTalk = <boolean>;** |

## 2.9.5 CONTROL LATCH MACROS

These macros are accessed by expanding the 'Clearcom' > 'ScriptLibrary' > 'ControlLatch' entry in the Available Modules menu.

| Macro | Description |
|---|---|
| CreateLatch() [ControlLatch] | Creates and returns a control latch e.g. **ControlLatch AlwaysOff = ControlLatch.Cre- ateLatch()** |
| Equals(Object) [bool] | Tests the equivalence of two objects and returns True or False. e.g. **bool <result> = <object1>.equals<(object2)>;** |
| GetHashCode() [int] | Returns the hash code of an object previously created by a control as an integer. e.g. **int <result> = <object>.GetHashCode();** |
| GetType() [Type] | Returns the type of an object previously created by a control macro. e.g. **Type <result> = <object>.GetType();** |

| Macro | Description |
|---|---|
| ResetsWhenOff(ControlLatch) [void] | This control latch function will reset a latch when a con- trol input is off. The example below shows sequence to get a control reference, create a latch and set the latch to reset when the control input is off.<br>**ControlMacro INPUT1 = ControlMacro.GetCon- trol("INB", "");**<br>**ControlLatch LATCH1 = ControlLatch.CreateLatch(); LATCH1.ResetsWhenOff(INPUT1);** |
| ResetsWhenOn(ControlLatch) [void] | This control latch function will reset a latch when a con- trol input is on. The example below shows sequence to get a control reference, create a latch and set the latch to reset when the control input is on.<br>**ControlMacro INPUT1 = ControlMacro.GetCon- trol("INB", "");**<br>**ControlLatch LATCH1 = ControlLatch.CreateLatch(); LATCH1.ResetsWhenOn(INPUT1);** |
| SetsWhenOff(ControlLatch) [void] | This control latch function will set a latch when a control input is off. The example below shows sequence to get a control reference, create a latch and assign the latch to be set when the control input is off.<br>**ControlMacro INPUT1 = ControlMacro.GetCon- trol("INB", "");**<br>**ControlLatch LATCH1 = ControlLatch.CreateLatch(); LATCH1.SetsWhenOff(INPUT1);** |
| SetsWhenOn(ControlLatch) [void] | This control latch function will set a latch when a control input is on. The example below shows sequence to get a control reference, create a latch and assign the latch to be set when the control input is on.<br>**ControlMacro INPUT1 = ControlMacro.GetCon- trol("INB", "");**<br>**ControlLatch LATCH1 = ControlLatch.CreateLatch(); LATCH1.SetsWhenOn(INPUT1);** |

| Macro | Description |
|---|---|
| TogglesWhenOff(ControlLatch) [void] | This control latch function will toggle a latch when a con- trol input is off. The example below shows sequence to get a control reference, create a latch and assign the latch to be toggled when the control input is off.<br>**ControlMacro INPUT1 = ControlMacro.GetCon- trol("INB", "");**<br>**ControlLatch LATCH1 = ControlLatch.CreateLatch(); LATCH1.TogglesWhenOff(INPUT1);** |
| TogglesWhenOn(ControlLatch) [void] | This control latch function will toggle a latch when a con- trol input is on. The example below shows sequence to get a control reference, create a latch and assign the latch to be toggled when the control input is on.<br>**ControlMacro INPUT1 = ControlMacro.GetCon- trol("INB", "");**<br>**ControlLatch LATCH1 = ControlLatch.CreateLatch(); LATCH1.TogglesWhenOn(INPUT1);** |
| ToString() [string] | Returns the string value of an object previously created by a control macro. e.g.<br>**string <result> = <object>.ToString();** |
| TriggersWhenOff(Control- Macro] [void] | Triggers a control macro object when the input condition is OFF e.g.<br>**<control macro>.TriggersWhenOff(ControlMacro);** |
| TriggersWhenOn(Control- Macro) [void] | This control latch function will trigger an output to the specified control when the input to the latch is on. The example below shows sequence to get a control refer- ence, create a latch and set the latch to trigger the con- trol when the input is on.<br>**ControlMacro OUTPUT1 = ControlMacro.GetCon- trol("OUTB", "");**<br>**ControlLatch LATCH1 = ControlLatch.CreateLatch(); LATCH1.TriggersWhenOn(OUTPUT1);** |

| Macro | Description |
|---|---|
| TriggersWhenOn(Action) [void] | This control latch function will trigger an output to the specified control when the input to the latch is on. The example below shows sequence to get a control refer- ence, create a latch and set the latch to trigger the con- trol when the input is on e.g. **ControlMacro OUTPUT1 = ControlMacro.GetCon- trol("OUTB", "");** **ControlLatch LATCH1 = ControlLatch.CreateLatch(); LATCH1.TriggersWhenOn(OUTPUT1);** |
| GetId [ushort] | Returns the ID of an object e.g. **ControlLatch <result> = <latch name>.GetId;** |
| TriggersWhenOff(Action) [void] | This control latch function will trigger an action on a crosspoint when the **ControlMacro OUTPUT1 = ControlMacro.GetCon- trol("OUTB", "");** **ControlLatch LATCH1 = ControlLatch.CreateLatch(); LATCH1.TriggersWhenOff(OUTPUT1);** |

## 2.9.6 CONTROL MACROS

Control macros act on system configuration objects to get or set parameters or to change the state of the object. The format of a control macro command is:

ControlMacro.<Macroname><parameters>;

The 'ControlMacro' command is used to create a copy of a system configuration object which can then to used in the control macro.

For example, the command:

ControlMacro GP19 = ControlMacro.GetControl("GP19");

will create a copy of the GPIO control object called GP19 created by EHX called GP19 which can be used in the control macro.

These macros are accessed by expanding the 'Clearcom' > 'ScriptLibrary' > 'ControlMacro' entry in the Available Modules menu.

| Macro | Description |
|---|---|
| CreateControl (string) [Control-Macro] | Creates a control named by the string e.g. **ControlMacro AND_60 = ControlMacro.CreateCon- trol("AND_60");** |
| CreateControl(string,bool) [Con-trolMacro] | Creates a control named by the string with a state set by the boolean e.g. **ControlMacro AND_60 = ControlMacro.CreateCon-trol("AND_60", true);** |
| Equals (Object) [bool] | Tests the equivalence of two objects and returns True or False. e.g. **bool <result> = <object1>.equals<(object2)>;** |
| GetAllEntities[(bool) [EntityOb-ject[]] | Returns a list of all known entities e.g. **EntityObject[] = ControlMacro.GetAll Entities(<local system only boolean>);** |
| GetAllPorts() [PortObject[]] | Returns all the known ports in a system to an allay of PortObject e.g. **PortObject[] allportsknown = ControlMacro.GetAll-Ports();** |
| GetAllStations() [PortObject[]] | Returns an array of PortObject containing all the panels in a system e.g. **PortObject[] stationsToCut = ControlMacro.GetAll-Stations();** |
| GetControl (Guid) [Control-Macro] | Gets the control information for the item named in the string parameter e.g. **ControlMacro.<object> = ControlMacro.GetCon-trol(<string>);** |
| GetControl (string) [Control-Macro] | Returns a reference to a control label with the given Talk label e.g. **ControlMacro CONTROL = ControlMacro.GetCon- trol("CTLA ")** |
| GetControl(string,string) [Con-trolMacro] | Returns a reference to a control label with the given Talk and Listen labels e.g. **ControlMacro CONTROL = ControlMacro.GetCon- trol("CTLA ", "     ")** |
| GetControl(string,string,int) [ControlMacro] | Returns a reference to a control label with the given Talk and Listen labels on the specified system e.g. **ControlMacro CONTROL = ControlMacro.GetCon- trol("CTLA ", "     ", 1)** |

| Macro | Description |
|---|---|
| GetEntities(string) [EntityOb- ject[]] | Return the entities specified in the string e.g. **EntityObject[] <entity> = ControlMacro.GetEnti- ties("<entity names>");** |
| GetEntity(string) [EntityObject] | Returns the entity specified in the string e.g. **EntityObject <entity> = ControlMacro.GetEn- tity(<entity name>);** |
| GetGroup(string) [EntityObject] | Returns the talk label for a group e.g. **EntityObject <entity> = ControlMacro.Get- Group("<talk label>");** |
| GetGroup(string string) [EntityO- bject] | Returns the talk and listen labels for the group specified as strings e.g.<br>**EntityObject <entity> = ControlMacro.get- Group("<talk label>", "<listen label>");** |
| GetGroup(string,string,int) [Enti- tyObject] | Returns the talk and listen labels for a group on the given system number e.g.<br>**EntityObject <entity> = ControlMacro.Get- Group("<talk label>, <listen label>, <system num- ber>);** |
| GetGroupMembers(EntityOb- ject) [EntityObject[]] | Returns the members of a specified group e.g. **EntityObject[] <entity> = ControlMacro.GetGroup- Members(<group identifier>);** |
| Macro | Description |
| GetHashCode () [int] | Returns the hash code of an object previously created by a control as an integer. e.g.<br>int <result> = <object>.GetHashCode(); |
| GetLocalSharedListenPortNum- berForRemoteEntity(PortOb- ject) [ushort] | Returns the port number on the remote system for the specified Listen port object e.g.<br>ushort <port number> = ControlMacro.GetLocal- SharedListenPortNumberForRemoteEntity(<port object>); |
| GetLocalSharedTalkPortNum- berForRemoteEntity(PortOb- ject) [ushort] | Returns the port number on the remote system for the specified Talk port object e.g.<br>ushort <port number> = ControlMacro.GetLocal- SharedTalkPortNumberForRemoteEntity(<port object>); |
| GetPartyLine(string) [EntityOb- ject] | Returns an entity for the named party line talk label e.g. EntityObject <entity> = ControlMacro.GetParty- Line(<party line name>); |

| Macro | Description |
|---|---|
| GetPartyLine(string string) [Enti- tyObject] | Returns the talk and listen labels for the party line e.g. EntityObject <entity> = ControlMacro.GetParty- Line(<talk label>, <listen label>); |
| GetPartylineMembers(EntityOb- ject) [EntityObject[]] | Returns the members of a party line e.g. EntityObject[] <entity> = ControlMacro.GetParty- LineMembers(EntityObject); |
| GetPort (Guid) [PortObject] | Gets the Guid for a port e.g.<br>PortObject <result> = ControlMacro.GetPort(<port- Guid>); |
| GetPort (int) [PortObject] | Returns a reference for the specified port number e.g.<br>PortObject p = ControlMacro.GetPort(600) |
| GetPort (string) [PortObject] | Returns a reference for a port with the given Talk label e.g.<br>PortObject p = ControlMacro.GetPort("Talk ") |
| GetPort(string string) [PortOb- ject] | Returns the reference for a port with the given Talk and Listen labels e.g.<br>PortObject p = ControlMacro.GetPort("Talk ", "Lstn ") |
| GetPort(string string int) [PortO- bject] | Returns the reference for a port with the given Talk and Listen labels and port number e.g.<br>PortObject p = ControlMacro.GetPort("Talk ", "Lstn ", 600) |
| GetAllPorts(bool) [PortObject[]] | Returns a list of port objects e.g.<br>**PortObject[] = ControlMacro.GetAllPorts(<local sys- tem only boolean>);** |

| Macro | Description |
|---|---|
| GetType () [Type] | Returns the type of an object previously created by a control macro. e.g.<br>**Type <result> = <object>.GetType();** |
| Inhibits(ControlMacro) [void] | Causes a specified control action to be inhibited e.g.<br>**<control identifier>.Inhibits(<control macro>);** |
| Inhibits(ControlMacro, ushort) [void] | Causes a specified control action to be inhibited e.g.<br>**<control identifier>.Inhibits(<control macro>,<logic parameter>);** |

| | |
|---|---|
| NameExists(string) [bool] | Returns a boolean indicating whether a named entity exists e.g. **bool <boolean> = ControlMacro.NameExists(<entity name>);** |
| Resets(ControlLatch) [void] | Resets the specified control latch e.g. **<entity instance>.Resets(latch name);** |
| SetDefaultGateway(string) [void] | Sets the default gateway address 1 on a frame to the specified string e.g. **ControlMacro.SetDefaultGateway(<default gateway address>);** |
| SetDefaultGateway2(string) [void] | Sets the default gateway address 2 on a frame to the specified string e.g. **ControlMacro.SetDefaultGateway2(<default gateway 2 address>);** |
| Sets(ControlLatch) [void] | Set a control latch e.g. **<control latch entity>.Sets(ControlLatch);** |
| SetSubnetMask(string) [void] | Sets the subnetmask 1 on a frame to the specified string e.g. **ControlMacro.SetSubnetMask(<subnet mask string>);** |
| SetSubnetMask2(string) [void] | Sets the subnetmask 2 on a frame to the specified string e.g. **ControlMacro.SetSubnetMask2(<subnet mask string>);** |
| Toggles(ControlLatch) [void] | Toggles the specified control latch e.g. **<latch instance>.Toggles(ControlLatch);** |
| Triggers (ControlMacro) [void] | This command executes a previously defined control. For example if an instance ‘GP23’ has been defined using the GetControl control macro and the control ‘fireLED1’ has been defined using the ControlActions macro the control ‘fireLED1 can be used as the parame- ter to the Trigger macro e.g. **GP23.Triggers (fireLED1);** |
| Triggers (Action) [void] | This command executes a previously defined action. For example if an instance ‘GP23’ has been defined using the GetControl control macro and the action ‘fireLED1’ has been defined using the ControlActions macro the action ‘fireLED1 can be used as the parame-ter to the Trigger macro.e.g. **GP23.Triggers (fireLED1);** |

| | |
|---|---|
| TriggersIf (ControlMacro, Condi- tion, ControlMacro) [void] | Triggers an action if the result of the condition test on the two control macros is met e.g. **<result>.TriggersIf(<control1>,<condition>,<con- trol2>);** |
| TriggersIf(CrosspointCon- trol,Condition,ControlMacro) [void] | Triggers a crosspoint control for a specified condition e.g. **<crosspoint>.TriggersIf(CrospointControl, <condi- tion>, ControlMacro);** |
| TriggersIf(CrosspointCon- trol,Condition,CrosspointCon- trol) [void] | Triggers a crosspoint control for a specified condition e.g. **<crosspoint>.TriggersIF(CrosspointControl, <condi- tion>, CrosspointControl);** |
| ControlObject [ControlObject] | Creates a control macro for an object e.g. **ControlMacro <result> = <object>.ControlObject;** |
| ToString () [string] | Returns the string value of an object previously created by a control macro. e.g. **string <result> = <object>.ToString();** |

## 2.9.7 CROSSPOINT CONTROL

| Macro | Description |
|---|---|
| Equals(Object) [bool] | Tests the equivalence of two objects and returns True or False. e.g. **bool <result> = <object1>.equals<(object2)>;** |
| GetDestination() [PortObject] | Returns the destination port for a crosspoint e.g. **PortObject <destination> = <port>.GetDestination;** |
| GetHashCode() [int] | Returns the hash code of an object previously created by a control as an integer. e.g. **int <result> = <object>.GetHashCode();** |
| GetSource() [PortObject] | Returns the source port for a crosspoint e.g. **PortObject <source> = <port>.GetSource;** |
| GetType() [Type] | Returns the type of an object previously created by a control macro. e.g. **Type <result> = <object>.GetType();** |
| Resets(ControlLatch) [void] | Resets a control latch object e.g. **<latch>.Resets(ControlLatch);** |

| | |
|---|---|
| Sets(ControlLatch) [void] | Sets a control latch object e.g.<br>**<latch>.Sets(ControlLatch);** |
| Toggles(ControlLatch) [void] | Toggles a control latch object e.g.<br>**<latch>.Toggles(ControlLatch);** |
| ToString() [string] | Returns the string value of an object previously created by a control macro. e.g.<br>**string <result> = <object>.ToString();** |
| Triggers(ControlMacro) [void] | Triggers a control macro from a latch e.g.<br>**<latch>.Triggers(ControlMacro)** |
| Triggers(Action] [void] | Triggers an action from a latch e.g.<br>**<latch>.Triggers(Action);** |
| On [bool] | Sets a crosspoint On/Off state to that specified by the boolean e.g.<br>**crosspointControl.On = <boolean>** |
| Priority [uint] | Sets a crosspoint priority level to the value specified e.g.<br>**crosspointControl.Priority = <priority>;** |

## 2.9.8 CURRENT MACROS

| Macro | Description |
|---|---|
| Equals(Object) [bool] | Tests the equivalence of two objects and returns True or False. e.g.<br>**bool <result> = <object1>.equals<(object2)>;** |
| GetHashCode() [int] | Returns the hash code of an object previously created by a control as an integer. e.g.<br>**int <result> = <object>.GetHashCode();** |
| IPAddress() [string) | Returns the IP address as a string e.g.<br>**string <result string> = Current.IPAddress();** |
| SystemNumber() [int] | Returns the system number as an integer e.g.<br>**int <result> = Current.systemNumber();** |
| ToString() [string] | Returns the string value of an object previously created by a control macro. e.g.<br>**string <result> = <object>.ToString();** |
| GetType() [Type] | Returns the type of an object previously created by a control macro. e.g.<br>**Type <result> = <object>.GetType();** |

## 2.9.9 LOGGING MACROS

These macros are accessed by expanding the 'Shared' > 'Logging' > 'Logger' entry in the Available Modules menu. Logging macros allow informatory, warning, error and fatal error messages to be output to a logging device.

| Macro | Description |
|---|---|
| Debug (Exception, IFormatPro-vider, string, Object[]) [void] | Creates a debug object to be sent to a logger e.g. **<logger>.Debug(<exception>,<format><string for- mat>,Object[<object>]);** |
| Debug (Exception, Object) [void] | Creates a debug object to be sent to the logger e.g. **<logger>.Debug(<exception>,<object>);** |
| Debug (Exception, string, Object[]) [void] | Creates a debug object to be sent to a logger e.g. **<logger>.Debug(<exception>,<string for-mat>,Object[<object>]);** |
| Debug (IformatProvider, string, Object[]) [void] | Creates a debug object to be sent to a logger e.g. **<logger>.Debug(<format><string for-mat>,Object[<object>]);** |
| Debug (Object) [void] | Creates a debug object to be sent to a logger e.g. **<logger>.Debug(<Object[<object>]);** |
| Debug (string, Object[]) [void] | Creates a debug object to be sent to a logger e.g. **<logger>.Debug(<string format>,Object[<object>]);** |
| DebugLow (Exception, IFormat-Provider, string, Object[]) [void] | Creates a debug object to be sent to a logger e.g. **<logger>.Debug(<exception>,<format><string for- mat>,Object[<object>]);** |
| DebugLow (Exception, Object) [void] | Creates a debug object to be sent to a logger e.g. **<logger>.DebugLow(<exception>,Object[<objec t>]);** |
| DebugLow (Exception, string, Object[]) [void] | Creates a debug object to be sent to a logger e.g. **<logger>.DebugLow(<exception>,<string for-mat>,Object[<object>]);** |
| DebugLow (IFormatProvider, string, Object[]) [void] | Creates a debug object to be sent to a logger e.g. **<logger>.DebugLow(<format><string for-mat>,Object[<object>]);** |
| DebugLow (Object) [void] | Creates a debug object to be sent to a logger e.g. **<logger>.DebugLow(<Object[<object>]);** |
| DebugLow (string, Object[]) [void] | Creates a debug object to be sent to a logger e.g. **<logger>.DebugLow(<string for-mat>,Object[<object>]);** |

| Macro | Description |
|---|---|
| Equals (Object) [bool] | Tests the equivalence of two objects and returns True or False. e.g:<br>**bool <result> = <object1>.equals<(object2)>;** |
| Error (Exception, IFormatPro- vider, string, Object[]) [void] | Creates a error object to be sent to a logger e.g.<br>**<logger>.Error(<exception>,<format><string for- mat>,Object[<object>]);** |
| Error (Exception, Object) [void] | Creates a error object to be sent to a logger e.g.<br>**<logger>.Error(<exception>,Object[<object>]);** |
| Error (Exception, string, Object[]) [void] | Creates a error object to be sent to a logger e.g.<br>**<logger>.Error(<exception>,<string for- mat>,Object[<object>]);** |
| Error (IFormatProvider, string, Object[]) [void] | Creates a error object to be sent to a logger e.g.<br>**<logger>.Error(<format><string for- mat>,Object[<object>]);** |
| Error (Object) [void] | Creates a error object to be sent to a logger e.g.<br>**<logger>.Error(Object[<object>]);** |
| Error (string, Object[]) [void] | Creates a error object to be sent to a logger e.g.<br>**<logger>.Error(<string format>,Object[<object>]);** |
| Fatal (Exception, IFormatPro- vider, string, Object[]) [void] | Creates a fatal error object to be sent to a logger e.g.<br>**<logger>.Fatal(<exception>,<format><string for- mat>,Object[<object>]);** |
| Fatal (Exception, Object) [void] | Creates a fatal error object to be sent to a logger e.g.<br>**<logger>.Fatal(<exception>,Object[<object>]);** |
| Fatal (Exception, string, Object[]) [void] | Creates a fatal error object to be sent to a logger e.g.<br>**<logger>.Fatal(<exception>,<string for- mat>,Object[<object>]);** |
| Fatal (IFormatProvider, string, Object[]) [void] | Creates a fatal error object to be sent to a logger e.g.<br>**<logger>.Fatal(<format><string for- mat>,Object[<object>]);** |
| Fatal (Object) [void] | Creates a fatal error object to be sent to a logger e.g.<br>**<logger>.Fatal(Object[<object>]);** |
| Fatal (string Object[]) [void] | Creates a fatal error object to be sent to a logger e.g.<br>**<logger>.Fatal(<string format>,Object[<object>]);** |
| GetLogger (string) [Logger] | Gets information for the logger specified in the string parameter e.g.<br>**Logger <result> = Logger.GetLogger(<string>);** |

| Macro | Description |
|---|---|
| GetLogger (Type) [Logger] | Gets information for the logger specified in the type parameter e.g.<br>**Logger <result> = Logger.GetLogger(<type>);** |
| GetType () [Type] | Returns the type of an object previously created by a control macro. e.g.<br>**Type <result> = <object>.GetType();** |
| HasLoggingStarted () [bool] | Returns a boolean to indicate whether logging has been started or not e.g.<br>**bool <result> = Logger.HasLoggingStarted();** |
| Info (Exception, IFormatPro- vider, string, Object[]) [void] | Creates a information object to be sent to a logger e.g.<br>**<logger>.Info(<exception>,<format><string for- mat>,Object[<object>]);** |
| Info (Exception, Object) [void] | Creates a information object to be sent to a logger e.g.<br>**<logger>.Info(<exception>,Object[<object>]);** |
| Info (Exception, string, Object[]) [void] | Creates a information object to be sent to a logger e.g.<br>**<logger>.Info(<exception>,<string for- mat>,Object[<object>]);** |
| Info (IFormatProvider, string, Object[]) [void] | Creates a information object to be sent to a logger e.g.<br>**<logger>.Info(<format><string for- mat>,Object[<object>]);** |
| Info (Object) [void] | Creates a information object to be sent to a logger e.g.<br>**<logger>.Info(Object[<object>]);** |
| Info (string, Object[]) [void] | Creates a information object to be sent to a logger e.g.<br>**<logger>.Info(<string format>,Object[<object>]);** |
| Push (string) [IDisposable] | creates a temporary string object e.g.<br>**IDisposable <object> = Logger.Push(<string>);** |
| StartLogging () [void] | Command to start logging e.g.<br>**Logger.StartLogging();** |
| ToString () [string] | Returns the string value of an object previously created by a control macro. e.g.<br>**string <result> = <object>.ToString();** |
| Warn (Exception, IFormatPro- vider, string, Object[]) [void] | Creates a warning object to be sent to a logger e.g.<br>**<logger>.Warn(<exception>,<format><string for- mat>,Object[<object>]);** |
| Warn (Exception, Object) [void] | Creates a warning object to be sent to a logger e.g.<br>**<logger>.Warn(<exception>,Object[<object>]);** |

| Macro | Description |
|---|---|
| Warn (Exception, string, Object[]) [void] | Creates a warning object to be sent to a logger e.g. **<logger>.Warn(<exception>,<string for-mat>,Object[<object>]);** |
| Warn (IFormatprovider, string, Object[]) [void] | Creates a warning object to be sent to a logger e.g. **<logger>.Warn(<format>,<string for-mat>,Object[<object>]);** |
| Warn (Object) [void] | Creates a warning object to be sent to a logger e.g. **<logger>.Warn(Object[<object>]);** |
| Warn (string, Object[]) [void] | Creates a warning object to be sent to a logger e.g. **<logger>.Warn(<string format>,Object[<object>]);** |
| IsDebugEnabled [bool] | Returns a boolean TRUE or FALSE indicating whether debug mode is enabled e.g. **Logger isDebugEnabled = <object>.IsDebugEna-bled;** |
| IsDebugLowEnabled [bool] | Returns a boolean TRUE or FALSE indicating whether debug low mode is enabled e.g. **Logger isDebugLowEnabled = <object>.IsDebugLo- wEnabled;** |
| IsErrorEnabled [bool] | Returns a boolean TRUE or FALSE indicating whether error reporting mode is enabled e.g. **Logger isErrorEnabled = <object>.IsErrorEnabled;** |
| IsFatalEnabled [bool] | Returns a boolean TRUE or FALSE indicating whether fatal error reporting mode is enabled e.g. **Logger isFatalEnabled = <object>.IsFatalEnabled;** |
| IsInfoEnabled [bool] | Returns a boolean TRUE or FALSE indicating whether info mode is enabled e.g. **Logger isInfoEnabled = <object>.IsInfoEnabled;** |
| IsWarnEnabled [bool] | Returns a boolean TRUE or FALSE indicating whether warn mode is enabled e.g. **Logger isWarnEnabled = <object>.IsEWarnEnabled;** |
| GetHashCode () [int] | Returns the hash code of an object previously created by a control as an integer. e.g: **int <result> = <object>.GetHashCode();** |

# 3 APPENDIX B EXAMPLE CONTROL MACROS

## 3.1 ACTIVATE SPECIFIC KEY LED

```
// When control LED0 is activated fourth key on each panel is illuminated red.
using System;

using ClearCom.ScriptHost;

using ClearCom.ScriptLibrary; using ClearCom.Entities; using EMS.MapClient;

using EMS.MapClient.Tables;

using EMS.MapClient.Tables.Actions; using Shared.Enums;


namespace CustomControlMacros

{

public class CustomMacro : ScriptBase

{

public override void OnUserStart()

{

// Fetch the control that will trigger this action. ControlMacro LED0 =
ControlMacro.GetControl("LED0");


// Fetch the panels we wish to activate the LED on. PortObject ISTA =
ControlMacro.GetPort("ISTA"); PortObject D4222 =
ControlMacro.GetPort("D4222");

PortObject[] panelArray = new PortObject[] { ISTA, D4222 }; foreach (PortObject
panel in panelArray)

{

// Set up LED indications.

// Note1: LED will only indicate if a key is assigned here, i.e. can't illuminate
empty key.

// Note2: Key numbers are silly, some start from 1, some from 0 etc

Action fireLed1 = ControlActions.ActivateLED(panel, 4, 1, 0,
Shared.Enums.LedRate.On, Shared.Enums.LedIndication.Red);


// Activate LEDs on. LED0.Triggers(fireLed1);

}
```

}

}

}

## 3.2 ACTIVATE LED ON ALL KEYS TO DESTINATION

// When control LED1 is activated, any key on any panel to I2003 is illuminated red. using System;

using ClearCom.ScriptHost;

using ClearCom.ScriptLibrary; using ClearCom.Entities; using EMS.MapClient;

using EMS.MapClient.Tables;

using EMS.MapClient.Tables.Actions; using Shared.Enums;

namespace CustomControlMacros

{

public class CustomMacro : ScriptBase

{

public override void OnUserStart()

{

// Fetch the elements we need.

ControlMacro LED1 = ControlMacro.GetControl("LED1"); PortObject I2003 = ControlMacro.GetPort("I2003");

// Set up LED indications.

Action fireLed1 = ControlActions.ActivateLED(I2003, Shared.Enums.LedRate.On, Shared.Enums.LedIndication.Red);

// Activate LEDs on. LED1.Triggers(fireLed1);

}

}

}

## 3.3 TRIGGER ACTION WHEN BOTH A AND B ARE SET

// When control AND1 is activated and control AND2 is activated, activate control FRLY1 using System;

```
using ClearCom.ScriptHost;
using ClearCom.ScriptLibrary; using ClearCom.Entities; using EMS.MapClient;
using EMS.MapClient.Tables;
using EMS.MapClient.Tables.Actions; using Shared.Enums;

namespace CustomControlMacros
{
public class CustomMacro : ScriptBase
{
public override void OnUserStart()
{

// Fetch the elements we need.
ControlMacro AND1 = ControlMacro.GetControl("AND1"); ControlMacro AND2 =
ControlMacro.GetControl("AND2"); ControlMacro FRLY1 =
ControlMacro.GetControl("FRLY1");

FRLY1.TriggersIf(AND1, Condition.AND, AND2);
}
}
}
```

## 3.4 TRIGGER ACTION WHEN ALL OF A AND B AND C ARE SET

```
// When control A1 is activated AND control A2 is activated AND control A3 is
activated, activate control FRLY4


using System;
using ClearCom.ScriptHost; using ClearCom.ScriptLibrary; using
ClearCom.Entities; using EMS.MapClient;
using EMS.MapClient.Tables;
using EMS.MapClient.Tables.Actions; using Shared.Enums;

namespace CustomControlMacros
```

```
{
public class CustomMacro : ScriptBase
{
public override void OnUserStart()
{

// Fetch the elements we need.
ControlMacro A1 = ControlMacro.GetControl("A1"); ControlMacro A2 =
ControlMacro.GetControl("A2"); ControlMacro A3 =
ControlMacro.GetControl("A3"); ControlMacro FRLY4 =
ControlMacro.GetControl("FRLY4");

// Note that each control can only have one TriggersIf, so create an intermediate
control
// to test the first 2 inputs.
ControlMacro intermediate = ControlMacro.CreateControl("IMDTE", true);
intermediate.TriggersIf(A1, Condition.AND, A2);
FRLY4.TriggersIf(intermediate, Condition.AND, A3);
}
}
}
```

## 3.5 CUT TALK TO STUDIO

```
// When control ST-CT is activated, prevents all panels from talking to port I2003.
using System;
using ClearCom.ScriptHost;
using ClearCom.ScriptLibrary; using ClearCom.Entities; using EMS.MapClient;
using EMS.MapClient.Tables;
using EMS.MapClient.Tables.Actions; using Shared.Enums;

namespace CustomControlMacros
{
public class CustomMacro : ScriptBase
{
public override void OnUserStart()
{

ControlMacro STCT = ControlMacro.GetControl("ST-CT");  PortObject[]
stationsToCut = ControlMacro.GetAllStations();

PortObject STUD1 = ControlMacro.GetPort("I2003"); foreach (PortObject station
in stationsToCut)

{

Action rOff1 = ControlActions.RouteOff(station.PortNumber, STUD1.PortNumber,
0);  STCT.Triggers(rOff1);

}

}

}

}
```

## 3.6 CUT TALK TO STUDIO, EXCLUDING SOME PANELS

```
// When control ST-CT is activated, prevents all panels apart from ISTA and I2003
from talking to port I2003

using System;

using ClearCom.ScriptHost; using ClearCom.ScriptLibrary; using
ClearCom.Entities; using EMS.MapClient;

using EMS.MapClient.Tables;

using EMS.MapClient.Tables.Actions; using Shared.Enums;


namespace CustomControlMacros

{

public class CustomMacro : ScriptBase

{

public override void OnUserStart()

{


ControlMacro STCT = ControlMacro.GetControl("ST-CT");  PortObject[]
stationsToCut = ControlMacro.GetAllStations();

PortObject STUD1 = ControlMacro.GetPort("I2003"); foreach (PortObject station
in stationsToCut)

{
```

```
if (Exclude(station)) continue;

Action rOff1 = ControlActions.RouteOff(station.PortNumber, STUD1.PortNumber,
0);  STCT.Triggers(rOff1);
}
}

private bool Exclude(PortObject station)
{
if (station.TalkLabel.Trim() == "ISTA") return true;

if (station.TalkLabel.Trim() == "I2003") return true;

return false;
}
}
}
```

## 3.7 TRIGGER ACTION WHEN BOTH A IS SET AND A CROSSPOINT IS MADE

```
// When control AND1 is activated AND ISTA talks to D4222, activate control
FRLY2 using System;
using ClearCom.ScriptHost;
using ClearCom.ScriptLibrary; using ClearCom.Entities; using EMS.MapClient;
using EMS.MapClient.Tables;
using EMS.MapClient.Tables.Actions; using Shared.Enums;

namespace CustomControlMacros
{
public class CustomMacro : ScriptBase
{
public override void OnUserStart()
{
// Fetch the elements we need.
```

ControlMacro AND1 = ControlMacro.GetControl("AND1"); ControlMacro FRLY2 = ControlMacro.GetControl("FRLY2");

// Fetch the panels we wish to get the crosspoint between. PortObject ISTA = ControlMacro.GetPort("ISTA"); PortObject D4222 = ControlMacro.GetPort("D4222");

// Create the control that will be triggered on the crosspoint. CrosspointControl crosspointControl = new CrosspointControl(ISTA, D4222);

FRLY2.TriggersIf(crosspointControl, Condition.AND, AND1);

}

}

}

## 3.8 TRIGGER ACTION WHEN GROUP 1 MEMBER TALKS TO GROUP 2 MEMBER

// If any panel in group 1 talks to a panel in group 2, control "FRLY3" is activated using System;

using ClearCom.ScriptHost;

using ClearCom.ScriptLibrary; using ClearCom.Entities; using EMS.MapClient;

using EMS.MapClient.Tables;

using EMS.MapClient.Tables.Actions; using Shared.Enums;

namespace CustomControlMacros

{

public class CustomMacro : ScriptBase

{

public override void OnUserStart()

{

PortObject[] allStations = ControlMacro.GetAllStations(); ControlMacro FRLY3 = ControlMacro.GetControl("FRLY3");

// Test each panel to see if it is in group 1

foreach (PortObject possibleGroup1Station in allStations)

```
{
if (!IsInGroup1(possibleGroup1Station)) continue;

// Test each panel to see if it is in group 2
foreach (PortObject possibleGroup2Station in allStations)
{
if (!IsInGroup2(possibleGroup2Station)) continue;

// We have a pair of panels, one from group1, one from group2

// Create the control that will be triggered on the crosspoint.
CrosspointControl crosspointControl = new
CrosspointControl(possibleGroup1Station, possibleGroup2Station);
crosspointControl.Triggers(FRLY3);
}
}
}

private bool IsInGroup1(PortObject station)
{
if (station.ListenAlias.Contains("*")) return true;

return false;
}

private bool IsInGroup2(PortObject station)
{
if (station.ListenAlias.Contains("#")) return true;

return false;
}

}
}
```

## 3.9 HEADSET-SELECT ON

```
// When control HS-ON is activated forces headset-select on for panel D4222
using System;
using ClearCom.ScriptHost;
using ClearCom.ScriptLibrary; using ClearCom.Entities; using EMS.MapClient;
using EMS.MapClient.Tables;
using EMS.MapClient.Tables.Actions; using Shared.Enums;

namespace CustomControlMacros
{
public class CustomMacro : ScriptBase
{
public override void OnUserStart()
{
// When control HS-ON is activated forces headset-select on for panel D4222

// Fetch the elements we need.
ControlMacro HSON = ControlMacro.GetControl("HS-ON"); PortObject D4222 =
ControlMacro.GetPort("D4222");

HSON.Triggers(ControlActions.HeadsetSelect(D4222));
}
}
}
```

## 3.10 HEADSET-SELECT ON ALWAYS

```
// Forces headset-select on for panel D4222 using System;
using ClearCom.ScriptHost;
using ClearCom.ScriptLibrary; using ClearCom.Entities; using EMS.MapClient;
using EMS.MapClient.Tables;
using EMS.MapClient.Tables.Actions; using Shared.Enums;

namespace CustomControlMacros
```

```
{
public class CustomMacro : ScriptBase
{
public override void OnUserStart()
{
PortObject D4222 = ControlMacro.GetPort("D4222");

// Create dummy crosspoint, and turn it on
CrosspointControl crosspointControl = new CrosspointControl(1022, 1022);
crosspointControl.On = true;

// Make the always-on crosspoint trigger the headset-select action
crosspointControl.Triggers(ControlActions.HeadsetSelect(D4222));

}
}
}
```

## 3.11 LOUDSPEAKER-CUT ON

```
// When control LS-CT is activated forces loudspeaker cut on for panel D4222
using System;
using ClearCom.ScriptHost;
using ClearCom.ScriptLibrary; using ClearCom.Entities; using EMS.MapClient;
using EMS.MapClient.Tables;
using EMS.MapClient.Tables.Actions; using Shared.Enums;

namespace CustomControlMacros
{
public class CustomMacro : ScriptBase
{
public override void OnUserStart()
{
// Fetch the elements we need.
ControlMacro HSON = ControlMacro.GetControl("LS-CT"); PortObject D4222 =
ControlMacro.GetPort("D4222");

HSON.Triggers(ControlActions.CutLoudspeaker(D4222));
}
}
}
```

## 3.12 IFB CONTROL MACROS

You can use the following control macros to to edit virtual IFBs.

**Note:** Some of the macros can only be used if you enable the following setting in the EHX configuration software: For any port configured as a panel, select **Port Properties** > **Global Options**, and then select the **IFB** checkbox.

- Add caller (IFB checkbox enabled)
- Add caller (IFB checkbox not enabled)
- Add destination
- Add return
- Add source (IFB checkbox enabled)
- Add source (IFB checkbox not enabled)

### 3.12.1 ADD CALLER (IFB CHECKBOX ENABLED)

```
public static Action RouteToIfbOn(PortObject source, PortObject destination);
//add caller
```

### 3.12.2 ADD CALLER (IFB CHECKBOX NOT ENABLED)

```
public static Action RouteToIfbOn(PortObject source, ushort ifb); //add caller
```

### 3.12.3 ADD DESTINATION

```
public static Action DestinationToIfbOn(PortObject source, PortObject
destination); //add destination
```

### 3.12.4 ADD RETURN

```
public static Action ReturnToIfbOn(PortObject source, ushort ifb); //add return
```

## 3.12.5 ADD SOURCE (IFB CHECKBOX ENABLED)

public static Action SourceToIfbOn(PortObject source, PortObject destination); //add source

## 3.12.6 ADD SOURCE (IFB CHECKBOX NOT ENABLED)

public static Action SourceToIfbOn(PortObject source, ushort ifb); //add source

## 3.12.7 EXAMPLE OF USING IFB CONTROL MACROS

Here is an example of how you might use IFB control macros.

```
using System;

using ClearCom.ScriptHost;

using ClearCom.ScriptLibrary;

using ClearCom.Entities;

using EMS.MapClient;

using EMS.MapClient.Tables;

using Shared.Enums;

using Action = EMS.MapClient.Tables.Actions.Action;


namespace CustomScript

{

public class UserScript : ScriptBase

{

public override void OnUserStart()

{

//This macro will allow a caller port (caller) to talk to an IFB port (ifb port) when a
control (do this) is fired

//get a control action from the physical system which will fire the actions

ControlMacro do_this = ControlMacro.GetControl("do th", "is   ");


//make a refererence to a port which has the 'IFB' tick box set (so behaves as a
global IFB)

PortObject ifb_port = ControlMacro.GetPort("ifb p", "ort  ");

//make a reference for a 'caller' port

PortObject caller = ControlMacro.GetPort("calle", "r    ");
//make an action called 'fire' which lets the caller call the IFB port
Action Fire = ControlActions.RouteToIfbOn(caller, ifb_port);
//when you see the control happen on the system, trigger the action to fire the
IFB
do_this.Triggers(Fire);
}
}
}
```

# 4 APPENDIX C KEY NUMBERING ON PANELS

This appendix gives the key numbering for the various panel types to enable control macros to be written to control specific keys on different types of panels.

Panels use region 1 unless otherwise stated. Keys on the main page are on page 0; keys on subsequent pages are on pages 1, 2, 3 etc.

The key numbers for panels are given below.

4212

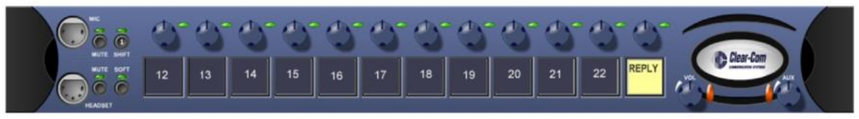


**Figure 4-1 4212 Panel Keys**

4215

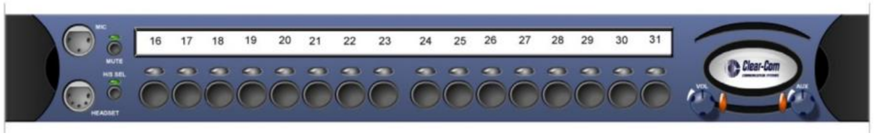


**Figure 4-2…4215 Panel Keys**

4222

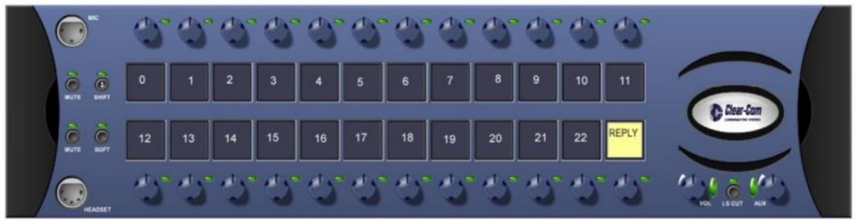


**Figure 4-3.. 4222 Panel Keys**

4224

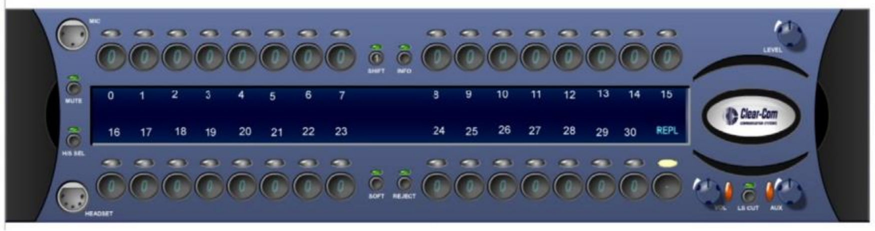


**Figure 4-4.. 4224 Panel Keys**

4226

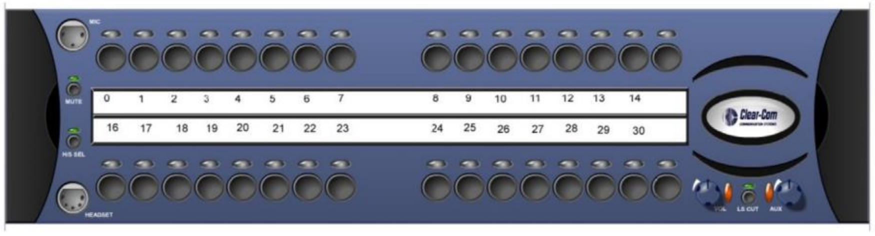


**Figure 4-5.. 4226 Panel Keys**

I-Station

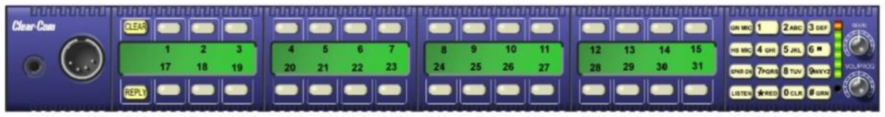


**Figure 4-6.. i-Station Panel Keys**

ICS-1008

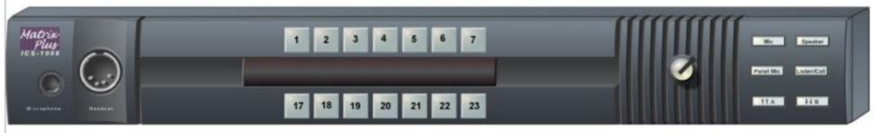


**Figure 4-7.. ICS-1008 Panel Keys**

ICS-1016

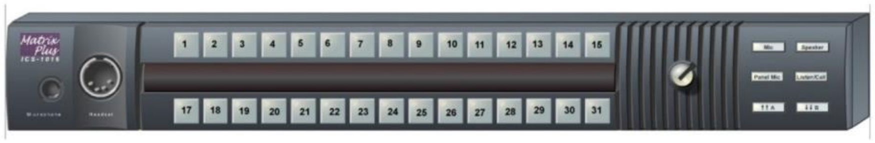


**Figure 4-8.. ICS-1016 Panel Keys**

ICS-102

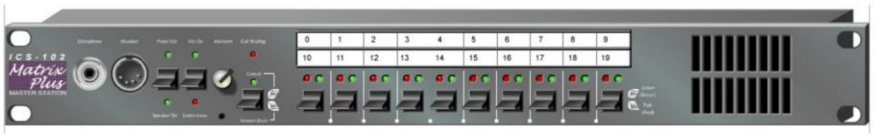


**Figure 4-9.. ICS-102 Panel Keys**

ICS-2003

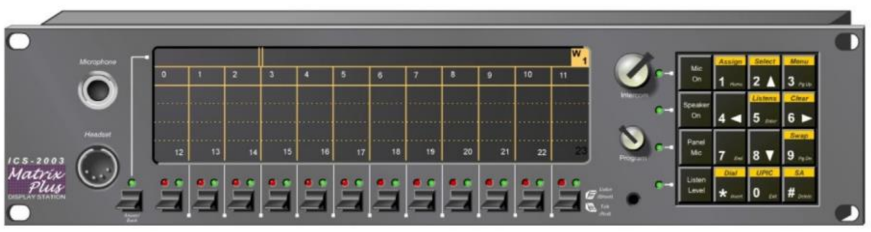


Figure 4-10.. ICS-2003 Panel Keys

ICS-52

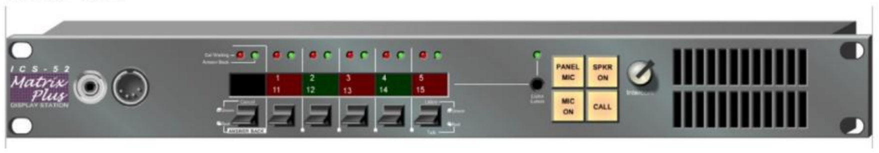


Figure 4-11 ICS-52 Panel keys

ICS-62

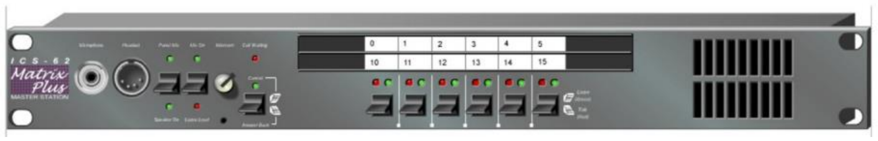


Figure 4-12...ICS-62 Panel Keys

ICS-92

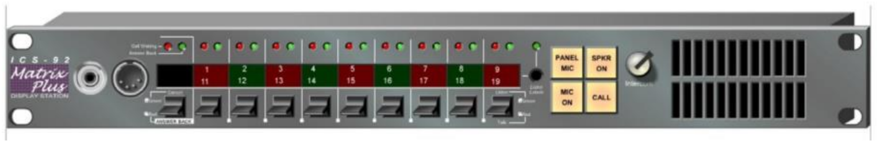


Figure 4-13.. ICS-92 Panel Keys

V 1RU Lever Key

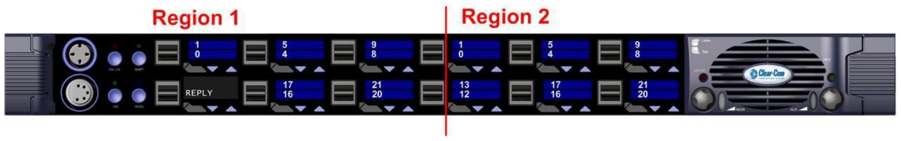


Figure 4-14.. V12LD Panel Keys

V 1RU Pushbutton

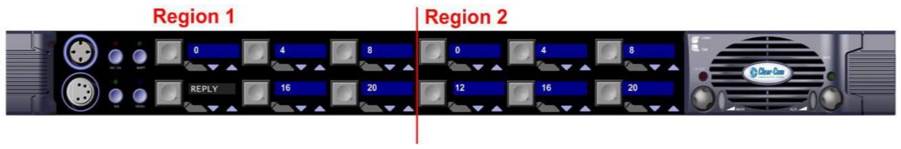


Figure 4-15.. V12PD Panel Keys

V 1RU Rotary

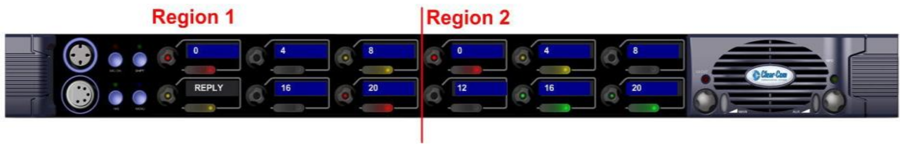


Figure 4-16.. V12RD Panel Keys

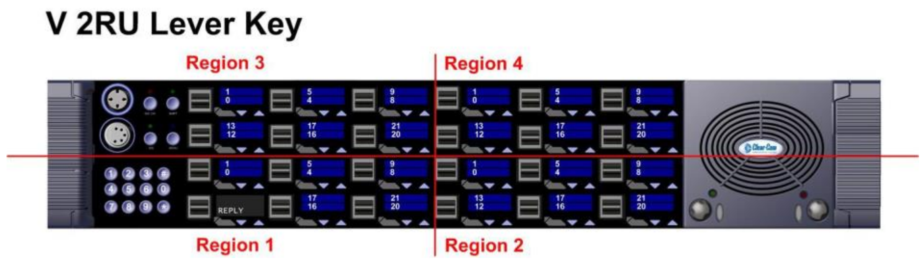


Figure 4-17.. V24LD Panel Keys

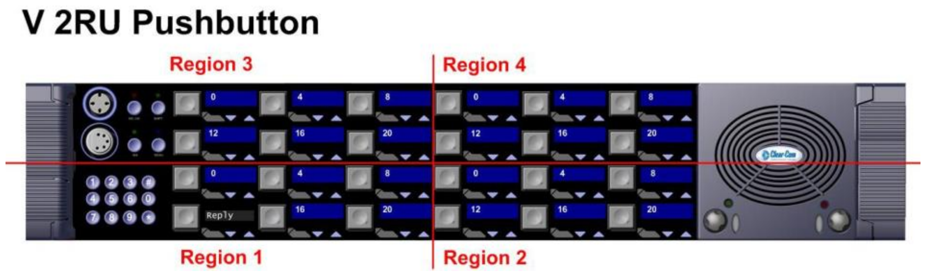


Figure 4-18.. V24PD Panel Keys

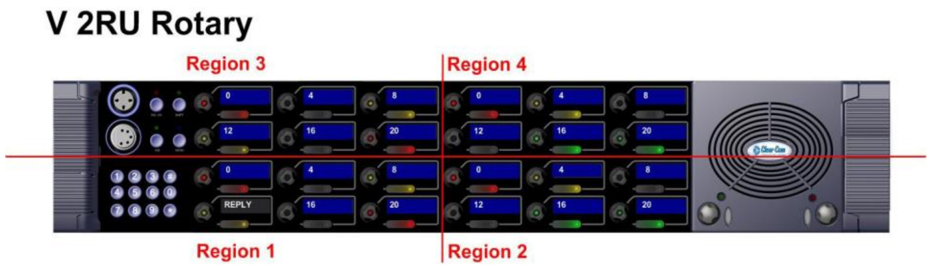


Figure 4-19.. V24RD Panel keys

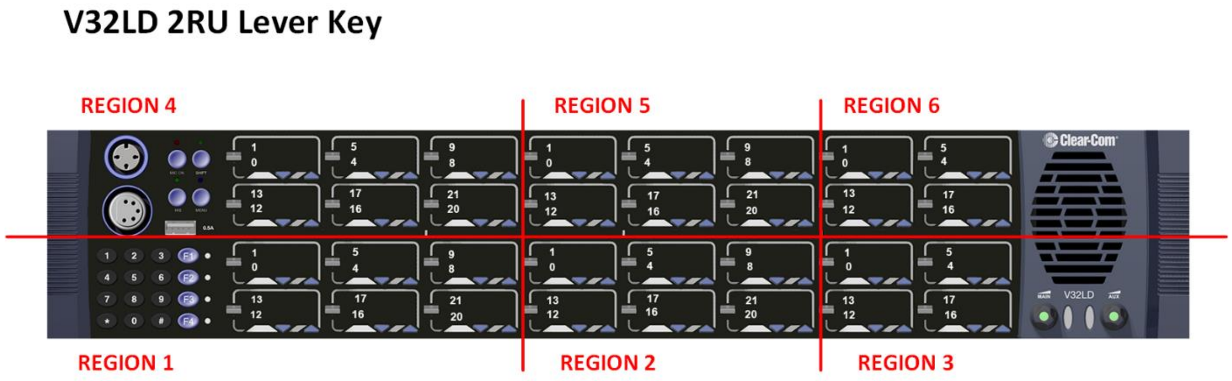


Figure 4-20 V32LD Panel keys

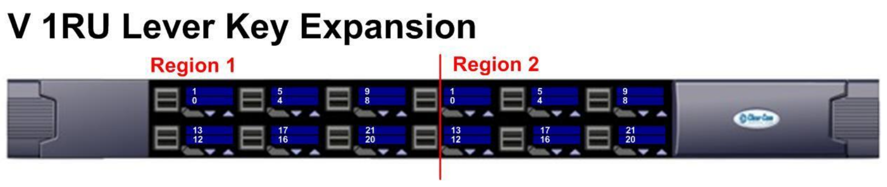


**Figure 4-21.. V12LDE Panel Keys**

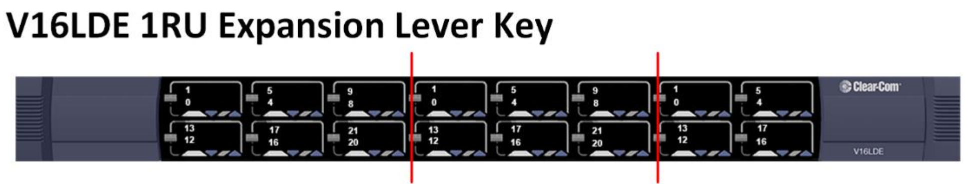


**Figure 4-22   V16LDE Panel Keys**

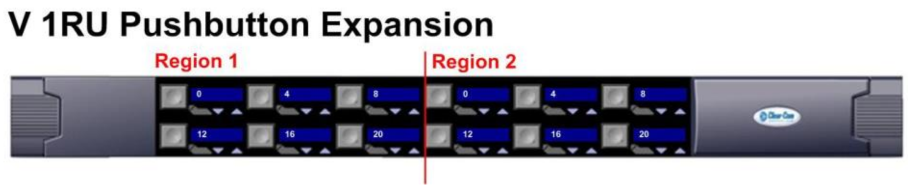


**Figure 4-23.. V12PDE Panel keys**

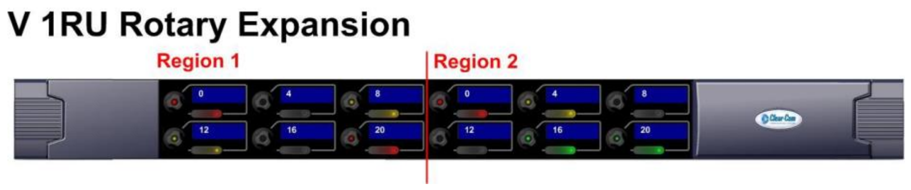


**Figure 4-24.. V12RDE Panel keys**

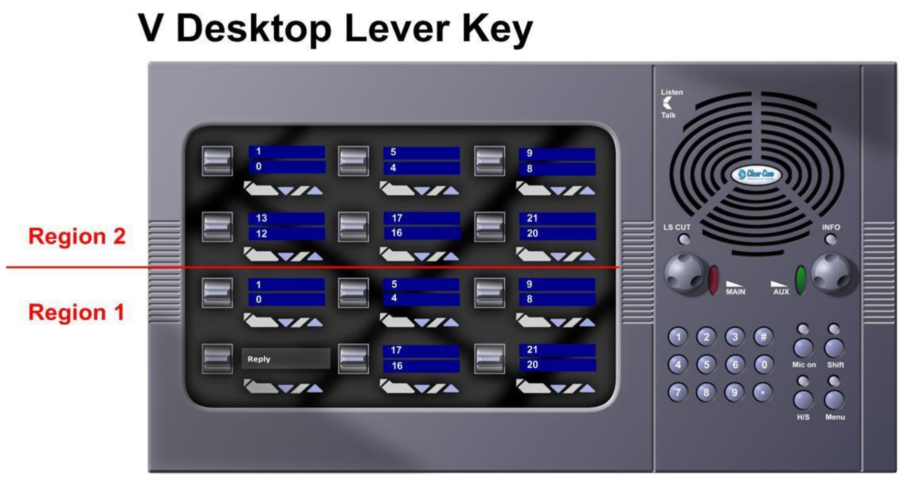


**Figure 4-25.. V12LDD Panel Keys**

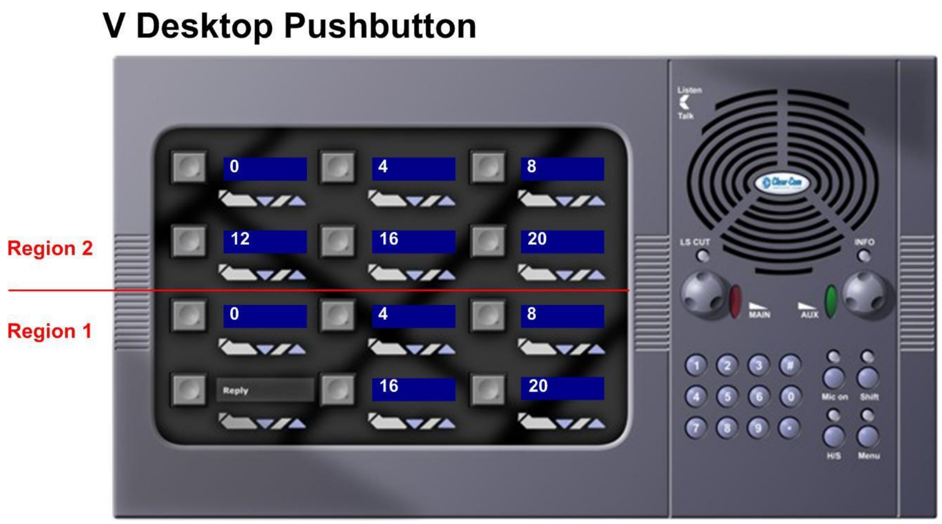


**Figure 4-26.. V12PDD Panel Keys**

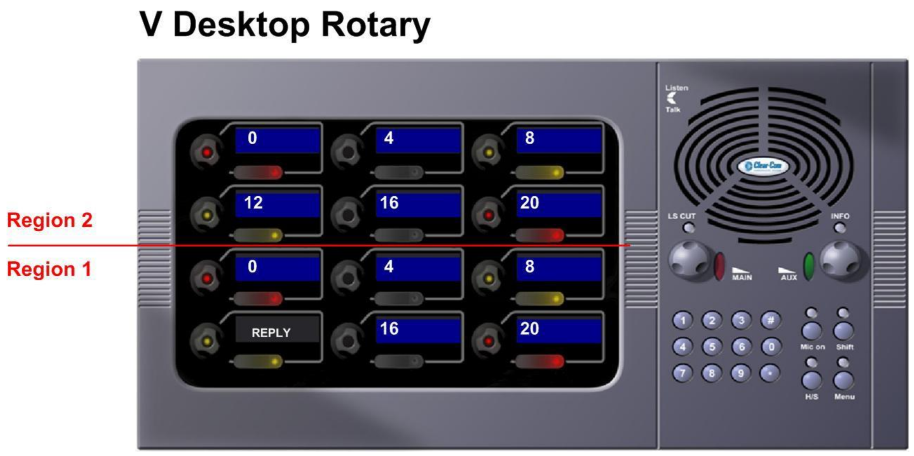


**Figure 4-27.. V12RDD Panel Keys**

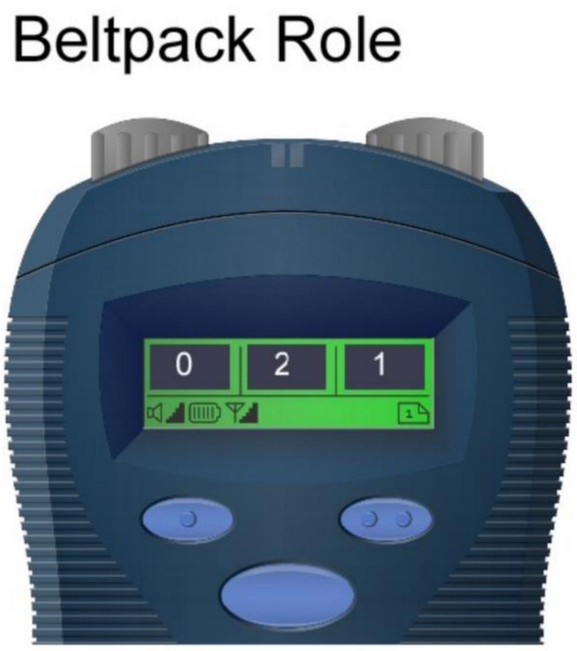


Figure 4-28.. FreeSpeak Beltpack Keys

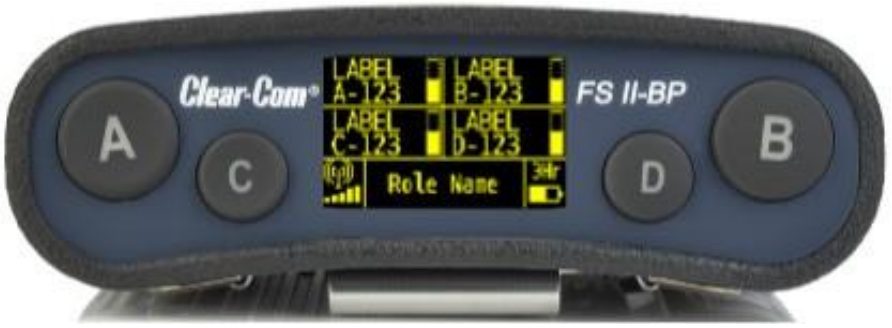


Figure 4-29.. FreeSpeak II Beltpack Keys

# 5 GLOSSARY

**Analog Port** Any of the Eclipse matrix's analog input/output RJ-45 connectors that are used to connect cable from the matrix to panels and interfaces. Each "port" connects to a separate audio channel in the matrix intercom system.

**Alias label** A label that is temporarily assigned and replaces a previously labeled port or conference.

**Bus** A bus is the channel or path between the components in the matrix along which electrical signals flow to carry information from one component to the next. In the Eclipse matrix the bus is located in the etched surface of the midplane.

**Call Signal** A call signal is an electronic signal sent from one panel or interface to another. A call signal can be audible and/or visual.

Typically a call signal is sent to get the attention of a panel operator who may have turned down their intercom speaker's volume or removed their headset. It can also be sent to activate an electronic relay.

**Canvas** The assignment area of Dynam-EC which can have any user labeled background.

**Category-5 cable** EIA/TIA 568 category specification relating to network cabling. Shielded category-5 cabling is required for Eclipse matrix wiring.

**CellCom** Digital wireless communications product. Sold under the CellCom name in USA and as FreeSpeak in Europe and Asia.

**Central Matrix** The term "central matrix" is used to differentiate the central hardware and software of the intercom system from the connected audio devices. The central matrix consists of:

1. The metal housing for the circuit cards and power supplies.
2. The circuit cards.

3. The power supplies.
4. The rear panel connectors which connect the matrix's hardware to panels and interfaces.

**Conference** An internal matrix virtual party line or busbar where many panels and interfaces can talk onto or listen from the party line without talking to themselves.

**Destination** A device such as an intercom panel, beltpack, or interface to which audio signals are sent. The device from which audio signals are sent is called a "source".

**Duplex** All real-time communication between individuals talking face to face is full duplex, meaning that they can both talk and listen simultaneously. The Eclipse matrices provide full-duplex audio.

**EHX** Eclipse Configuration Software. Software program that guides the operation of the central matrix circuit cards and connected panels.

**Ethernet** International standard which describes how information is transmitted across a network. Provides for the efficient organization of network components.

**Fiber-optic Cable** A fiber-optic cable consists of a glass core covered with a reflective material called "cladding" and several layers of buffer coating to protect the cable from the environment. A laser sends light pulses through the glass core to the other end of the cable.

**FreeSpeak** Digital wireless communications product. Sold under the FreeSpeak name in Europe and Asia and CellCom name in USA.

**FreeSpeak II** Advanced digital wireless communications product.

**Full Duplex** Refers to transmission of signals in two directions simultaneously.

**IFB** "Interruptible Foldback". The term "foldback" refers to sending "program" audio, or some other audio mix, back to announcers while they are on the air. Doing so allows announcers to monitor themselves, other announcers, videotapes of commercials, or some mix of sources, while they on the air. This is typically found in television news and live broadcast events.

Announcers typically wear a small ear piece so they can hear the selected foldback audio mix. When a director wants to give directions to an announcer on air, or to announce changes in the program, the director must “interrupt” the foldback. To do this, the director uses a channel specifically set up to interrupt the foldback audio.

**Interface Module** A piece of electronic hardware designed to convert the 4-wire signals of a central matrix port to some other form of communication, such as 2-wire party line, telephone, etc. The interface module is connected to a central matrix port. The external non-4-wire device is then connected to the interface module.

**ISO** The ISO function, short for “panel ISOlation”, allows a panel operator to call a destination and interrupt all of that destination’s other audio paths and establish a private conversation. When the call is completed the destination’s audio pathways are restored to their original state before the interruption.

**KeyGroup** KeyGroups provide a way of assigning a label to multiple panels simultaneously even within a networked matrix system. Once the KeyGroups have been defined using EHX, all the keys within a KeyGroup can be changed with a single assignment in Dynam-EC (Pro mode only).

**Label** A label is an alphanumeric name of up to five characters that identifies a source, destination, or control function accessed by an intercom panel. Labels appear in the displays of the intercom panel.

Labels can identify panels, ports interfaced to other external equipment, fixed groups, party lines, and special control functions.

**Multiplexing** The process by which two or more signals are transmitted over a single communications channel. Examples include time division and wavelength division multiplexing.

**Non-volatile Memory** Data stored in the CPU's firmware (ROM) that is not lost when the power is turned off.

**Palette** The port, keyGroup and Monitor selection screen in Dynam-EC.

**Panel** Also referred to as "station" in some cases (usually older manuals). Any intelligent intercom device connected to the rear-panel analog ports of the central matrix. This term does not refer to devices connected through interface modules.

**Party Line** A wired shared communication system based on a single screened pair of wires. See the Encore range. Matrix requires the CCI-22 to interface to it.

**Port** Any of the input/output connections (RJ-45 connectors) on the back panel of the central matrix. These connectors and the attached cables connect the central matrix to remote intercom devices. The term "port" emphasizes that the connection is a "portal" between the central matrix and the remote intercom devices.

**Program** Any separate audio source that is fed into the intercom channels. In television applications, for example, "program" audio is the audio that is broadcast on air.

**Rack Unit or RU** Standardized unit of mounting space on a rack panel. Each rack unit is 1.75 inches (44.45 mm) of vertical mounting space. Therefore 1 RU is 1.75 inches (44.45 mm) of vertical mounting space, 2 RU is 3.5 inches (88.9 mm), 3 RU is 5.25 inches (133.35 mm), and so on.

**Remote Panel** Any intelligent intercom device connected to the back-panel ports of the central matrix. This term does not refer to devices connected through interfaces.

**Sidetone** The sound of the panel operator's own voice heard in their own earphone as they speak.

**Source** In this manual, the term "source" refers to a device—such as an intercom panel, interface, or beltpack —that sends audio into the matrix. The device to which audio is sent is called a “destination”.

**VOX** In the Eclipse system, when audio at a panel exceeds a threshold, a light switches on at the panel’s port card to visually cue the operator. The threshold level is set in the Eclipse Configuration Software.

**V-Series** Communications panels used with Eclipse systems providing advanced facilities. Available in rack mount and desktop formats.

# 6 LIMITED WARRANTY

Vitec Group Communications (VGC) warrants that at the time of purchase, the equipment supplied complies with any specification in the order confirmation when used under normal conditions, and is free from defects in workmanship and materials during the warranty period.

During the warranty period VGC, or any service company authorized by VGC, will in a commercially reasonable time remedy defects in materials, design, and workmanship free of charge by repairing, or should VGC in its discretion deem it necessary, replacing the product in accordance with this limited warranty. In no event will VGC be responsible for incidental, consequential, or special loss or damage, however caused.

## 6.1 WARRANTY PERIOD

The product may consist of several parts, each covered by a different warranty period. The warranty periods are:

- Cables, accessories, components, and consumable items have a limited warranty of 90 days.
- Headsets, handsets, microphones, and spare parts have a limited warranty of one year.
- UHF wireless IFB products have a limited warranty of one year.
- UHF wireless intercom systems have a limited warranty of three years.
- All other Clear-Com and Drake brand systems and products, including beltpacks, have a limited warranty of two years.

The warranty starts at the time of the product's original purchase. The warranty start date for contracts which include installation and commissioning will commence from the earlier of date of the Site Acceptance Test or three months from purchase.

## 6.2 TECHNICAL SUPPORT

To ensure complete and timely support to its customers, VGC's User Support Center is staffed by qualified technical personnel. Telephone and email technical support is offered worldwide by the User Support Center.

The User Support Center is available to VGC's customers during the full course of their warranty period.

Instructions for reaching VGC's User Support Centers are given below.

Telephone for Europe, Middle East and Africa: +49 40 6688 4040 or

+44 1223 815000

Telephone for the Americas and Asia: +1 510 337 6600

Email: vitec.support@AVC.de

Once the standard warranty period has expired, the User Support Center will continue to provide telephone support if you have purchased an Extended Warranty.

For latest contact information please refer to the Service and Support section at www.clearcom.com.

## 6.3 WARRANTY REPAIRS AND RETURNS

Before returning equipment for repair, contact a User Support Center to obtain a Return Material Authorization (RMA). VGC representatives will give you instructions and addresses for returning your equipment. You must ship the equipment at your expense, and the support center will return the equipment at VGC's expense.

For out-of-box failures, use the following contact information:

Europe, Middle East and Africa

Tel: +44 1223 815000 Email: SalesSupportEMEA@vitecgroup.com

North America, Canada, Mexico, Caribbean & US Military

Tel: +1 510 337 6600 Email: SalesSupportUSA@vitecgroup.com

Asia Pacific & South America

Tel: +1 510 337 6600 Email: SalesSupportAPAC@vitecgroup.com

VGC has the right to inspect the equipment and/or installation or relevant packaging.

For latest contact information please refer to the Service and Support section at www.clearcom.com.

## 6.4 NON-WARRANTY REPAIRS AND RETURNS

For items not under warranty, you must obtain an RMA by contacting the User Support Center. VGC representatives will give you instructions and addresses for returning your equipment.

You must pay all charges to have the equipment shipped to the support center and returned to you, in addition to the costs of the repair.

## 6.5 EXTENDED WARRANTY

You can purchase an extended warranty at the time of purchase or at any time during the first two years of ownership of the product. The purchase of an extended warranty extends to five years the warranty of any product offered with a standard two-year warranty. The total warranty period will not extend beyond five years.

**Note:** VGC does not offer warranty extensions on UHF wireless intercom systems, or on any product with a 1-year or 90-day warranty.

## 6.6 LIABILITY

THE FOREGOING WARRANTY IS VGC'S SOLE AND EXCLUSIVE WARRANTY. THE IMPLIED WARRANTY OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE AND ANY OTHER REQUIRED IMPLIED WARRANTY SHALL EXPIRE AT THE END OF THE WARRANTY PERIOD. THERE ARE NO OTHER WARRANTIES (INCLUDING WITHOUT LIMITATION WARRANTIES FOR CONSUMABLES AND OTHER SUPPLIES) OF ANY NATURE WHATSOEVER, WHETHER ARISING IN CONTRACT, TORT, NEGLIGENCE OF ANY DEGREE, STRICT LIABILITY OR OTHERWISE, WITH RESPECT TO THE PRODUCTS OR ANY PART THEREOF DELIVERED HEREUNDER, OR FOR ANY DAMAGES AND/OR LOSSES (INCLUDING LOSS OF USE, REVENUE, AND/OR PROFITS). SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES OR THE LIMITATION ON HOW LONG AN IMPLIED WARRANTY LASTS, SO THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU. IN ANY EVENT, TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, VGC'S LIABILITY TO CUSTOMER HEREUNDER SHALL NOT UNDER ANY CIRCUMSTANCES EXCEED THE COST OF REPAIRING OR REPLACING ANY PART(S) FOUND TO BE DEFECTIVE WITHIN THE WARRANTY PERIOD AS AFORESAID.

This warranty does not cover any damage to a product resulting from cause other than part defect and malfunction. The VGC warranty does not cover any defect, malfunction, or failure caused beyond the control of VGC, including unreasonable or negligent operation, abuse, accident, failure to follow instructions in the manual, defective or improperly associated equipment, attempts at modification and repair not approved by VGC, and shipping damage. Products with their serial numbers removed or defaced are not covered by this warranty.

This warranty does not include defects arising from installation (when not performed by VGC), lightning, power outages and fluctuations, air conditioning failure, improper integration with non-approved components, defects or failures of customer furnished components resulting in damage to VGC provided product.

This limited warranty is not transferable and cannot be enforced by anyone other than the original consumer purchaser.

This warranty gives you specific legal rights and you may have other rights which vary from country to country.

# 7 TECHNICAL SUPPORT & REPAIR POLICY NOVEMBER 1, 2008

In order to ensure that your experience with Clear-Com and our World Class products is as beneficial, effective and efficient as possible, we would like to define the policies and share some "best practices" that can accelerate any problem solving processes which we may find necessary and to enhance your customer service experience. Our Technical Support, Return Material Authorization, and Repair Policies are set forth below. These Policies are subject to revision and constantly evolve in order to address our Customers' and the Market's needs. Accordingly these are provided by way of guidance and for information only and may be changed at anytime with or without Notice.

## TECHNICAL SUPPORT POLICY

a) Telephone, online, and e-mail technical support will be provided by the Customer Service Center free of charge during the Warranty Period.

b) Technical support will be provided free of charge for all software products under the following conditions:

   i) The application, operating, and embedded software is installed on a product covered by Clear-Com's Limited Warranty, and:

      (1) The software is at the current release level; or,

      (2) The software is one (1) version removed from current.

   ii) Older versions of software will receive "best-effort" support, but will not be updated to correct reported bugs or add requested functionality.

c) For Technical Support:

   i) North and South America, (inc. Canada, Mexico, and the Caribbean) & US Military: Hours: 0800 - 1700 Pacific Time

| | |
|---|---|
| Days: | Monday - Friday |
| Tel: | +1 510 337 6600 |
| Email: | CustomerServicesUS@clearcom.com |

   ii) Europe, the Middle East and Africa:

| | |
|---|---|
| Hours: | 0800 - 2000 Central European Time |
| Days: | Monday - Friday |
| Tel: | +49 40 853 999 700 |
| Email: | TechnicalSupportEMEA@clearcom.com |

iii) Asia-Pacific:

| | |
|---|---|
| Hours: | 0800 - 1700 Pacific Time |
| Days: | Monday - Friday |
| Tel: | +1 510 337 6600 |
| Email: | CustomerServicesAPAC@clearcom.com |

d) Email Technical Support is available for all Clear-Com branded products free of charge for the life of the product, or two years after a product has been classified as obsolete, whichever comes first.

e) Support for Distributor and Dealer Sales

i) Distributors and Dealers may utilize the Customer Service Centers once a system has been installed and commissioned. Clear-Com Systems and Applications Engineers will provide support to the Distributor from the pre-sales stage through to satisfactory installation for new system purchases. Customers will be encouraged to contact their Dealer or Distributor with their installation and technical support enquires rather than using the Customer Service Centers directly.

f) Support for Direct Sales

i) Customers may utilize the Customer Service Centers once a system has been installed and commissioned by Clear-Com Systems and Applications Engineers, or in the case of project installations, once the Project Team has completed the hand-over to the Support Centers.

## RETURN MATERIAL AUTHORIZATION POLICY

a) Authorizations: All products returned to Clear-Com or a Clear-Com Authorized Service Partner must be identified by a Return Material Authorization (RMA) number.

b) The Customer will be provided with an RMA number upon contacting Clear-Com Sales Support as instructed below.

c) The RMA number must be obtained from Clear-Com via phone or email prior to returning product to the Service Center. Product received by the Service Center without a proper RMA number is subject to return to the Customer at the Customer's expense.

d) Damaged equipment will be repaired at the Customer's expense.

e) Returns are subject to a 15% restocking fee.

f) Advance Warranty Replacements (AWRs);

i) *During the first 30 days of the Standard Warranty Period:* Once the equipment fault has been verified by Clear-Com or its authorized representative, Clear-Com will ship a new replacement product. The

Customer will be provided with an RMA number and be required to return the faulty equipment within 14 days of receipt of the replacement or will be invoiced for the list price of a new product.

ii) *During days 31-90 of the Standard Warranty Period:* Once the equipment fault has been verified by Clear-Com or its authorized representative, Clear-Com will ship a like-new, fully refurbished replacement product. The Customer will be provided with an RMA number and be required to return the faulty equipment within 14 days of receipt of the replacement or will be invoiced for the list price of a new product.

iii) To obtain an RMA number or request an AWR:

(1) North and South America, Asia-Pacific, and US Military: Hours: 0800 - 1700 Pacific Time

| | |
|---|---|
| Days: | Monday - Friday |
| Tel: | +1 510 337 6600 |
| Email: | SalesSupportUS@clearcom.com |

(2) Europe, the Middle East and Africa:

| | |
|---|---|
| Hours: | 0800 - 1700 GMT + 1 |
| Days: | Monday - Friday |
| Tel: | + 44 1223 815000 |
| Email: | SalesSupportEMEA@clearcom.com |

iv) Note: AWRs are not available for UHF WBS Analog wireless intercom systems. UHF WBS Analog wireless intercom systems out-of-box failures must be returned to Alameda for repair.

v) Note: Out-of-box failures returned after 90 days will be repaired and not replaced unless approved by Clear-Com Management.

vi) Note: AWRs are not available after 90 days of receipt of product unless an AWR Warranty Extension is purchased at the time of product purchase.

vii) Note: Shipping charges, including duties, taxes, and insurance (optional), to Clear-Com's factory is the responsibility of the Customer. Shipping AWRs from Clear-Com is at

Clear-Com's expense (normal ground or international economy delivery). Requests for expedited shipping (E.g. "Next-Day Air") and insurance are the responsibility of the Customer.

## REPAIR POLICY

a) Repair Authorizations: All products sent to Clear-Com or a Clear-Com Authorized Service Partner for repair must be identified by a Repair Authorization (RA) number (see above).

b) The Customer will be provided with an RA number upon contacting Clear-Com Customer Services as instructed below.

c) The RA number must be obtained from Clear-Com via phone or email prior to returning product to the Service Center. Product received by the Service Center without a proper RA number is subject to return to the Customer at the Customer's expense.

d) Return for Repair

i) Customers are required to ship equipment at their own cost (including transportation, packing, transit, insurance, taxes and duties) to Clear-Com's designated location for repair.

(1) Clear-Com will pay for the equipment to be returned to the Customer when it is repaired under warranty.

(2) Shipping from Clear-Com is normal ground delivery or international economy. Requests for expedited shipping (E.g. "Next-Day Air") and insurance are the responsibility of the Customer.

ii) **Clear-Com does not provide temporary replacement equipment ("loaner") during the period the product is at the factory for repair.** Customers should consider a potential prolonged outage during the repair cycle, and if required for continuous operations purchase minimum spare equipment required or purchase an AWR Warranty Extension.

iii) No individual parts or subassemblies will be provided under warranty, and warranty repairs will be completed only by Clear-Com or its Authorized Service Partners.

iv) Customers requesting a non-warranty repair will be provided an estimate of the total repair cost prior to the return of the equipment. In the event that Clear-Com is unable to estimate

the cost of repair, the Customer may elect to return the product to the factory for an estimate. The Customer is responsible for shipping costs both to and from the factory in the event they choose not to accept the estimate.

v) The Customer must provide either a purchase order for the repair work, or will be required to make an advance payment (as a debit against the Dealer's line of credit, or credit card) prior to the repaired product being returned to the Customer.

vi) For requesting a Repair Authorization number:

(1) North and South America, Asia-Pacific, and US Military: Hours: 0800 - 1700 Pacific Time

| | |
|---|---|
| Days: | Monday - Friday |
| Tel: | +1 510 337 6600 |
| Email: | CustomerServicesUS@clearcom.com |

(2) Europe, the Middle East and Africa:

| | |
|---|---|
| Hours: | 0800 - 2000 Central European Time |
| Days: | Monday - Friday |
| Tel: | +49 40 853 999 700 |
| Email: | TechnicalSupportEMEA@clearcom.com |

vii) Note: Clear-Com's Limited Warranty does not cover normal wear and tear. The Customer will be charged the full cost of the repair if their equipment has been tampered with by

non-approved personnel, or has been subject to damage through electrical failure, liquid damage or mishandling. The Customer Service Center will provide the Customer with a cost estimate for any such repairs prior to undertaking the work.