CRIMSON 3
USER MANUAL

Copyright © 2003-2013 Red Lion Controls Inc.

All Rights Reserved Worldwide.

The information contained herein is provided in good faith, but is subject to change without notice. It is supplied with no warranty whatsoever, and does not represent a commitment on the part of Red Lion Controls. Companies, names and data used as examples herein are fictitious unless otherwise stated. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, without the express written permission of Red Lion Controls Inc.

The Red Lion logo is a registered trademark of Red Lion Controls Inc.

Crimson and the Crimson logo are registered trademarks of Red Lion Controls Inc.

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

Written by Mike Granby and Jesse Benefiel.

TABLE OF CONTENTS

CRIMSON 3
USER MANUAL

GETTING STARTED

Welcome to Crimson 3—the latest version of Red Lion's widely-acclaimed operator interface configuration software. If you have used an earlier version of Crimson, you will soon notice that Crimson 3 provides a huge number of improvements—while retaining all of the power that you have grown used to. If you are new to Crimson, be sure to read at least the first few chapters of this manual to get an introduction to how the software works. Either way, you will soon find out how Crimson 3 makes it easier and quicker for you to design powerful and attractive operator interface solutions.

SYSTEM REQUIREMENTS

Crimson 3 is designed to run on any version of Microsoft Windows from Windows XP onwards. Memory requirements are modest, and any system that meets the minimum system requirements for its operating system will be capable of running Crimson without any problems. About 150MB of free disk space will be needed for installation, and you should ideally have a display with sufficient resolution to allow the editing of display pages without having to scroll too much. For a VGA target device, an XGVA PC is recommended.

INSTALLING THE SOFTWARE

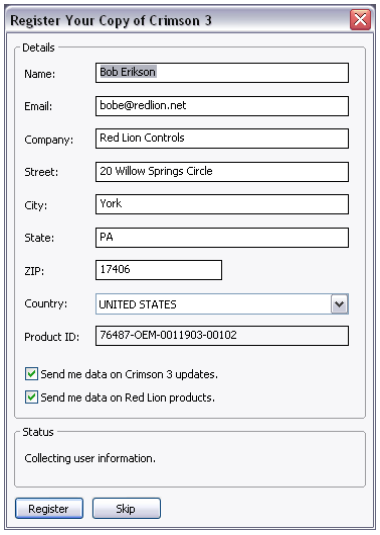
Crimson 3 is supplied as a Microsoft Installer package or an `msi` file. You will typically have downloaded this file from Red Lion's website, but if you have downloaded it from another source, please check that Windows is satisfied with the package's digital signature so that you are assured of receiving genuine Red Lion software…

As shown above, the publisher should show as Red Lion Controls Inc, and you should be able to click on the publisher's name to verify the integrity of the digital signature. Once you are happy with the package, press the Run button to start the installation.

The installation process is fairly standard, and ought to proceed without much interaction beyond your specifying the target directory. One the process complete, look at your Start Menu and find the Red Lion Controls folder. Click the Crimson 3 icon to start the software.

REGISTRATION

When you first run Crimson 3, you will be offered a chance to register your software…



While registration is optional, we strongly recommend that you take the opportunity to provide us with your contact details so that we can inform you of Crimson updates and of associated products. Since registration requires an Internet connection, you may skip the process if so do not have such a connection available. Crimson will periodically remind you if you are running an unregistered copy of the software.

CHECKING FOR UPDATES

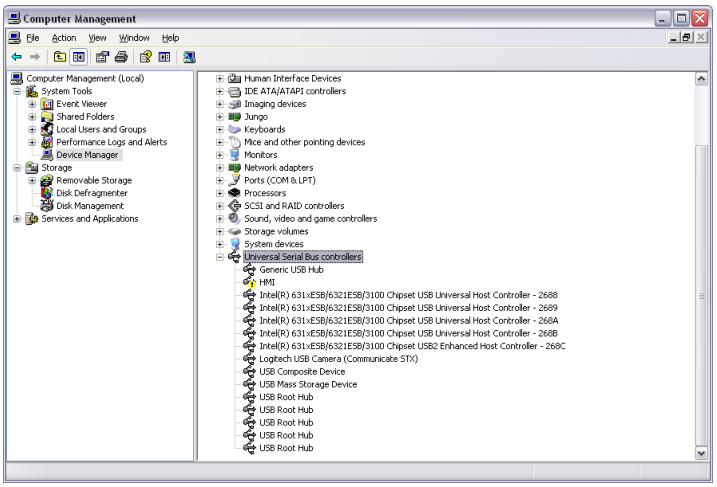
If you have an Internet connection, you can use the Check for Update command in the Help menu to scan Red Lion's web site for a new version of Crimson 3. If a later version than the one you are using is found, Crimson will ask if it should download the upgrade and update your software automatically. You may also manually download the upgrade from Red Lion's website by visiting the Downloads page within the Support section.

INSTALLING THE USB DRIVERS

If you've followed the instructions that came with your target hardware, you will not yet have connected the hardware to your PC. Now that you have completed the Crimson 3 installation, you may safely connect the device using a standard USB cable. After some churning, Windows should indicate that it has found the drivers for the new hardware and that it is ready for operation. No further user intervention should be required.

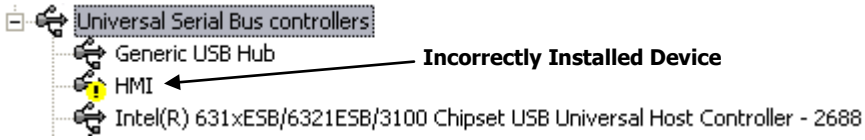
TROUBLESHOOTING

If you connected the target device to your PC before installing Crimson 3, it is possible that an aborted installation has made it impossible for you to install the drivers correctly. To check for this, open the Windows Device Manager by finding the My Computer icon, right-clicking and selecting the Manage command. A window similar to the one below should appear…

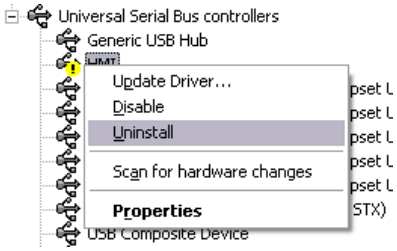


The exact process for getting to this point will vary from one operating system to another, but the basic idea is the same: Find the My Computer icon—either on the desktop or on your start menu—right-click it and select Manage. If that doesn't work, select the System option from the Control Panel, and activate the Device Manager from the Hardware tab.

If you have a problem with your USB drivers, you will see a yellow icon carrying an exclamation point under the Universal Serial Bus controllers category. The name of the icon may be HMI or Loader or something similar. The broken driver is shown in close-up below…



To fix the problem, right-click on the broken device and select Uninstall from the menu…



After asking for confirmation, Windows will remove the device from your system. You can now power the Crimson target device off. After a couple of seconds, reapply power and Windows will start the driver installation process once again.

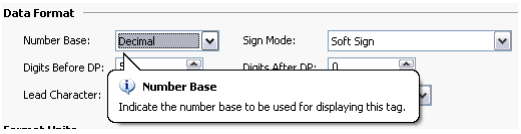
As mentioned above, Crimson actually uses distinct device drivers for the boot loader and for the Crimson runtime. You may thus have to repeat this repair process for each driver, although it is unlikely that things got beyond the boot loader if that install failed.

GETTING ASSISTANCE

If you hit a problem or need assistance, several resources are available.

BALLOON HELP

Crimson contains a very useful feature called Balloon Help…



This facility allows you to see help information for each item within Crimson. It is controlled via the icon at the right-hand edge of the toolbar or via options on the Help menu. The default mode allows the help text to be displayed by pressing the **F1** key, providing a quick way of getting information if you are unsure of the settings for a given field. Keep this in mind, and your life will be a lot easier!

TECHNICAL SUPPORT

Technical assistance is available on the web at…

http://www.redlion.net/support.html

Or, we can be contacted via email at…

support@redlion.net

You may also call +1 (717) 767-6511 and ask for the HMI Support Team.

ONLINE FORUMS

A number of online forums exist to support users of PLCs and HMIs. Red Lion recommends the Q&A forum at http://www.plctalk.net/qanda/. The discussion board is populated by many experts who are willing to help, and Red Lion's own technical support staff also keep an eye out for questions relating to our products.

THE NEXT STEPS

Since Crimson 3 has a completely new user interface, we suggest that existing Crimson users at least give the next chapter a cursory reading. We also suggest that you look at the chapters on tags and display page configuration, as some of the concepts used by Crimson 2 have been simplified, and many things can be achieved through easier methods. If you are completely new to Crimson, please try to read at least as far as the start of the chapter on Widgets.

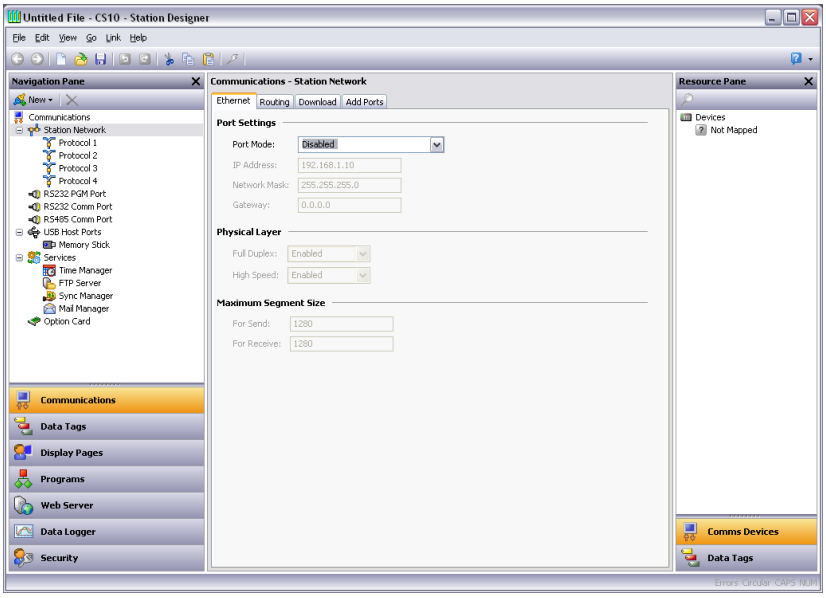
Good luck, and have fun!

CRIMSON BASICS

To run Crimson, select the Crimson 3 icon in the Red Lion Controls section of your Start Menu. After a couple of seconds, Crimson will appear. The first thing you will notice is the updated user interface that we have adopted. This new interface allows quicker navigation and faster database construction. We hope that it will greatly improve your productivity.

WINDOW LAYOUT

The main Crimson window comprises three sections…



THE NAVIGATION PANE

The left-hand portion of the window is called the Navigation Pane. It is used to move between different categories of items within a Crimson configuration file. Each category is represented by a bar at the base of the pane, and clicking on that bar will navigate to that section. The top section of the Navigation Pane shows the available items in the current category, and provides a toolbar to allow those items to be manipulated. If you want to make the top section larger, you can pick up and drag the dividing line between it and the category bars.

THE RESOURCE PANE

The right-hand portion of the window is called the Resource Pane. It is used to access various items that are of use when editing the current category. Just like the Navigation Pane, it contains a number of categories which can be accessed via the category bars. Items in a given resource category can be drag-and-dropped to the places where you wish to use them. For example, a data tag can be picked up from the Resource Pane and dropped on a configuration field to make that field dependant on the value of the selected tag. Many items can also be double-clicked, thereby setting the current field to that item.

THE EDITING PANE

The central portion of the window is used to edit the currently selected item. Depending on the selection, it may contain a number of tabs, each showing a given set of the properties for that item, or it may contain an editor specific to the item that you are working on.

COLLAPSING PANES

Either or both of the Navigation Pane and Resource Pane can be collapsed to the edge of the main window in order to free-up more space for the Editing Pane. To close a pane, click on the 'x' in the top left-hand corner of its title bar. It will then be replaced by a smaller bar marked with arrows. Clicking this bar will expand the associated pane. Clicking on the pushpin icon of a temporarily-expanded pane will lock it in place.

THE CATEGORIES

The main categories in a Crimson database are as follows…

COMMUNICATIONS

This category specifies which protocols are to be used on the target device's serial ports and Ethernet ports. Where master protocols are used (i.e. protocols in which the Red Lion hardware initiates data transfer to and from a remote device) you can also use this icon to specify one or more devices to be accessed. Where slave protocols are used (i.e. protocols in which the Red Lion hardware receives and responds to requests from other systems) you can specify which data items are to be exposed for read or write access. You can also use this category to move data between remote devices via the protocol converter, to configure expansion cards and to configure services.

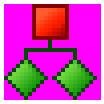
DATA TAGS

This category defines the data items that are to be used to be access data within the remote devices, or to store information within the target device. Each tag has a variety of properties, including formatting data, which specifies how the data held within a tag is to be shown on the device's display or in other contexts such as web pages. By specifying this information within the tag, Crimson removes the need for you to reenter formatting data each time a tag is displayed. More advanced tag properties include alarms that may activate when various conditions relating to the tag occur, or triggers, which perform programmable actions when those conditions are met.

DISPLAY PAGES

This category is used to create and edit display pages. The page editor allows you to display various graphical items known as primitives. These vary from simple items, such as rectangles and lines, to more complex items that can be tied to the value of a particular tag or to an expression. By default, such primitives use the formatting information defined when the tag was created, although this information can be overridden if required. You may also use the editor to specify what actions should be taken when keys or primitives are pressed, released or held down.

PROGRAMS

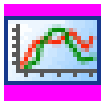


This category is used to create and edit programs using Crimson's unique C-like programming language. These programs can perform complex decision-making or data manipulation operations based upon data items within the system. They serve to extend the functionality of Crimson beyond that of the standard functions included in the software, thereby ensuring that even the most complex applications can be tackled. Programs can call upon a variety of system functions to perform common operations.

WEB SERVER

This category is used to configure Crimson's web server and to create and edit web pages. The web server is capable of providing remote access to the target device via a number of mechanisms. First, you can use Crimson to create automatic web pages which contain lists of tags, with each formatted according to the tag's properties. Second, you can create a custom site using a third party HTML editor such as Microsoft FrontPage, and then include special text to instruct Crimson to insert live tag values. Finally, you can enable Crimson's unique remote access and control feature, which allows a web browser to view the target device's display and control its keyboard. The web server can also be used to access CSV files from the Data Logger.

DATA LOGGER



This category is used to create and manage data logs, each of which can record any number of variables to the target device's CompactFlash card. Data may be recorded as quickly as once per second. The recorded values will be stored in CSV or Comma Separated Variable files that can easily be imported into applications such as Microsoft Excel. The files can be accessed by swapping-out the CompactFlash card, by mounting the card as a drive on a PC connected on the target device's USB port, or by accessing them via Crimson's web or FTP servers using an Ethernet port or a modem. Log files can be protected via cryptographic signatures to ensure that they have not been tampered with since they were written to CompactFlash.

SECURITY

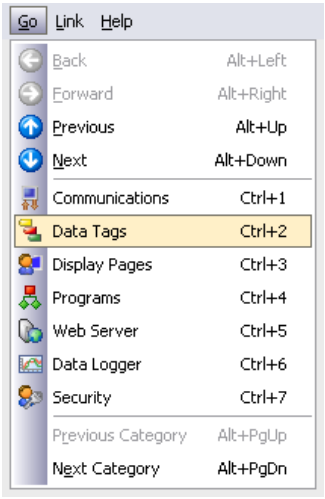
This category is used to create and manage the various users of the target device, as well as the access rights granted to each. Real names may also be given, which allows the security logger to record not only what data was changed and when, but also by whom. The rights required to modify a particular tag or to access a page are set via the security properties of the individual item. Rights can also be assigned to allow or deny access to the FTP server or the web server.

MODULES

This category is only available only when configuring a Modular Controller. It is used to define up to 16 IO modules to be connected to the Master, and to define their operation. Each module operates independently, and will continue to control the associated process even when the Master is being reprogrammed. Data from the modules can be mapped into data tags just as when working with other external devices.

GETTING AROUND

The easiest way to get around a Crimson database is to click on the category bars in the Navigation Pane, and then click on the item you want to edit. However, a number of shortcuts exist to allow quicker movement and thus greater productivity. Most of these shortcuts can be accessed via the Go menu, or via associated key combinations…



BACK AND FORWARD

The first icon on the toolbar or the **ALT+LEFT** key combination can be used to move back to items that you had previously selected. The next icon or the **ALT+RIGHT** key combination can then be used to move forward again, returning to the item you first started with. This facility is very useful when switching between database categories.

CATEGORY SHORTCUTS

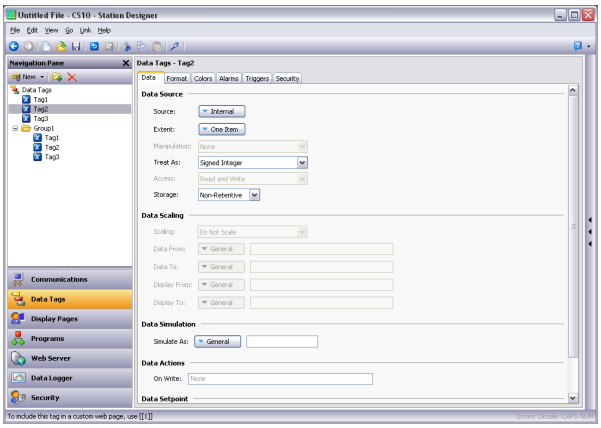
Each category is allocated a shortcut key sequence, comprising the **CTRL** key and a number indicating the category's position in the Navigation Pane. For example, the Communications section can be accessed directly by using the **CTRL+1** combination. You can also move up and down in the category list by using the **ALT+PGUP** and **ALT+PGDN** key combinations.

ITEM SHORTCUTS

If you are working in the Editing Pane, you can switch between items by using the **ALT+UP** and **ALT+DOWN** key combinations. Crimson will move to the previous or next item in the item list, and will try to keep the currently-selected data field the same. This is very useful if you want to change the same field on a number of items, as you do not have to keep navigating back to that field or switching to the Navigation Pane in order to change items.

NAVIGATION LISTS

Several categories in Crimson contain lists of items. For example, selecting the Data Tags category will cause the Navigation Pane to show a list of all the data tags in your database, allowing them to be selected and edited…



Items within these Navigation Lists can be manipulated in various ways…

- To quickly find an item, type the first few letters of its name. Crimson will select the first item that matches the characters you have entered. Typing more characters will make the selection more specific, while pressing **ESC** will allow a new sequence of search characters to be entered.

- To create an item, click on the New button in the Navigation Pane toolbar. For those lists that support only a single type of item, you may also use the **ALT+INS** key combination. The New button on the toolbar may offer a list of available items, allowing you to choose the type of the item you wish to create.

- To delete an item, either use the Delete icon in the Navigation Pane toolbar, or press the **ALT+DEL** key combination. If you delete a folder, all of the items within that folder will be deleted, too. Warnings are provided for multiple deletes, although they can always be reversed via the Undo command.

- To rename an item, select it and press **F2**. You may then enter the new name and press **ENTER**. Alternatively, select the item and then single-click on the name once more to activate editing. Again, press **ENTER** when you are finished.

WORKING WITH FOLDERS

Some lists support the grouping of items into folders. Folders can be created using the New Folder icon in the Navigation Pane toolbar, and can be renamed and deleted just like more conventional items. Creating an item with a folder selected will place that item in the selected folder. Folders can be nested up to any reasonable depth.

SORTING LISTS AND FOLDERS

An entire Navigation List or the contents of a folder may be sorted by right-clicking on the root item or the folder as appropriate, and selecting one of the Sort commands. Items may be

sorted in ascending or descending alphabetic order. Folders are always placed before other items, no matter which sort order is applied.

DRAG AND DROP OPERATIONS

Items in Navigation Lists can be drag-and-dropped within the list to change their position or to move them between folders. Holding down the **CTRL** key while dragging will result in a copy of the original item being created. The left-to-right position of an item may sometimes be used to decide where to place an item in the folder hierarchy. If the item is being dropped into the wrong folder, try moving left or right to get to the correct position.

Database items—such as tags, display pages or anything else—may also be dragged between database files by opening two copies of Crimson and dragging the items in question from the source database's Navigation Pane to that of the target database. If the appropriate category in the target is not already selected, temporarily holding the item that is being dragged over the required category bar for a second or so will select that category, thereby avoiding the need to abort and repeat the drag operation.

SEARCHING IN LISTS

While the shortcut detailed above is useful for jumping directly to a single item, you may sometimes want to find all the items that have names containing a particular string. This can be accomplished using the Find Item command shown on the Navigation Pane's toolbar. This command will search the current list, and place all the matching items in the Global Search Results List. You can step through this list using the **F4** and **SHIFT+F4** key combinations, or display the list in its entirety by pressing **F8**. For more information on the global search functions, refer to the section later in this chapter.

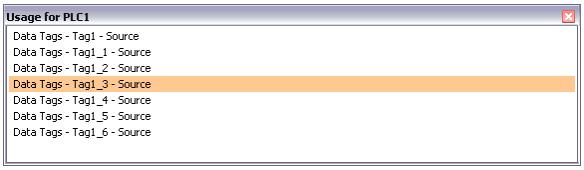
UNDO AND REDO

Crimson 3 implements a universal undo and redo structure. This means that you can load a database, edit it for hours, and then return it to its original state by simply holding down the **CTRL+Z** key combination. You can then re-apply your changes by holding down **CTRL+Y**. All your actions are remembered, and Crimson will navigate between items and categories automatically when reversing or re-implementing changes.

GLOBAL SEARCHING

Crimson provides several options for searching within a database. At the simplest level, you can search for a text string anywhere in the database by pressing the **CTRL+SHFIT+F** key combination. Alternatively, as you will see later, you may search for expressions which contain errors, or for items that reference a tag or a communications device. All of these search operations place their output in the Global Search Result List, allowing you to review the results, or to navigate back and forth between the items that have been located.

The results list can be displayed at any time by pressing the **F8** key…



The title bar of the window describes the search operation that produced the list, while each line contains the description an item that matched the search criteria. In the example above, right-clicking on a communications device and selecting the Find Usage command listed all the locations where the device was referenced. Double-clicking a given entry will jump directly to that item, while the **F4** and **SHIFT+F4** key combinations can be used to step back and forth through the list. The commands associated with this feature may also be accessed via the Find Global commands on the Edit menu.

WORKING WITH DATABASES

Crimson stores all the information about a particular configuration in what is called a database file. These files have the extension of `cd3`, although Windows Explorer will hide this extension if it is left in its default configuration. While Crimson 3 databases are still essentially text files, they are compressed and therefore cannot be directly edited using a text editor like Notepad. As you would expect, databases are manipulated via the commands found on the File menu. Most of these commands are standard for all Windows applications, and need no further explanation.

DATABASE IDENTIFIERS

Each database created by Crimson is given a unique identifier. This identifier is used upon download of a new database to determine if the target device should clear its internal memory and delete any log files recorded to CompactFlash. If the identifier matches that of the database already in the device, the database is assumed to be merely a different version of the same file, so the data is retained. Conversely, if the identifiers are different, the data is cleared. When you use the Save As command on the File menu to save a copy of a database file, Crimson will ask if you want to allocate a new identifier. Select Yes if this is going to be a new project, and select No if you are just saving a backup copy of what is essentially the same database. This will ensure that the target device's retentive data is cleared or preserved as is appropriate.

SAVING AN IMAGE

A Crimson-specific item on the File menu is Save Image. This command allows the creation of a file that can subsequently be used to update the database in a terminal via CompactFlash or optionally via a USB memory stick. The file contains a non-editable form of the database, plus any firmware and boot loader updates required for execution. Placing an image file called `image.ci3` in the root directory of the target device's CompactFlash card and then resetting the device will update the boot loader, firmware and database using the image file contents. Note that image files can optionally contain upload information, thereby allowing an editable version of the database file to be extracted from a terminal.

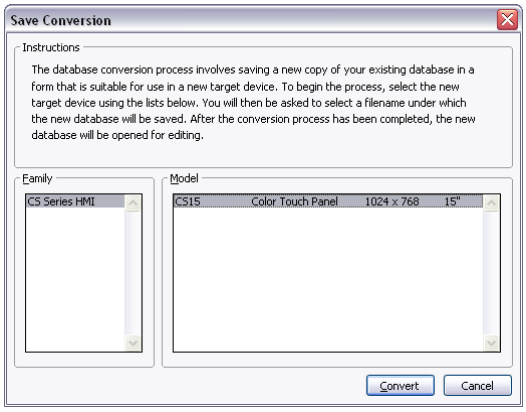
DATABASE PROTECTION

Databases can be password-protected using the Protection command on the File menu…

The Default Access parameter is used to define what level of access will be permitted without first entering the database password. A setting of Read-Only Access will allow the database to be opened, but will not allow changes to be made or saved. A setting of No Access will prevent all access without the password. The default setting of Full Access will allow the database to be opened for editing without any password being entered. Lost passwords can be recovered by Red Lion Controls for free, or for a nominal fee if you make a habit of it! Note that for security reasons, the recovered password will only ever be sent to the Recovery Email configured in the protection dialog box. Be sure to set this to a valid email address.

CONVERTING A DATABASE

A database designed for one target device may be converted for use on another by using the Save Conversion command on the File menu. The conversions that can be performed depend on the original target device, but most combinations are supported.



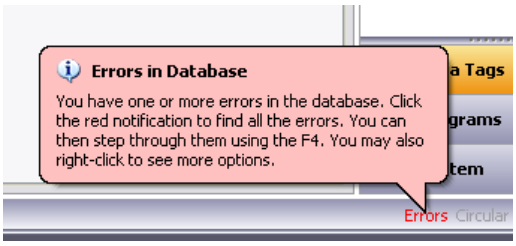
The conversion process is started by selecting the new target device using the dialog shown above. You will then be prompted for a new filename, and the converted database will be saved to disk. So as to avoid accidental destruction of existing databases, you may not convert a database without saving it under a new name. Once the converted database has been saved, it will be opened for editing and review.

The conversion process resizes any display pages to fit the new display format, and remaps communications devices to the appropriate ports on the new device based upon whether they use the RS-232 or RS-485 physical layer. It may not be possible to convert a database in its

entirety if, for example, the new device has fewer communications port than the original. You may thus have to perform a few adjustments after the conversion.

FINDING DATABASE ERRORS

Certain operations may produce errors in your database. For example, you may delete a communications device, or you may set a tag equal to an expression based on itself, thereby producing a circular reference. Crimson will warn you about any such errors by means of a red balloon that will appear above the status bar…



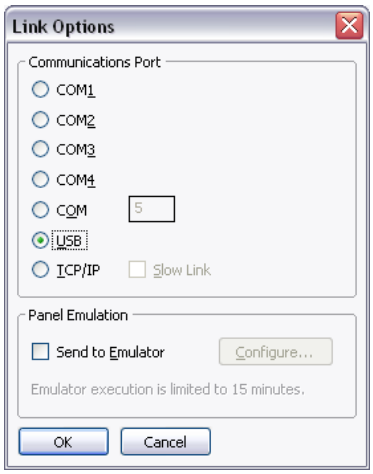
The balloon will fade after a few seconds, but the red indication in the status bar will remain to remind you of the error condition. Clicking on the indicator will search for all errors or circular references, and place them on the Global Search Results List so that you may review them using the standard **F4** and **SHIFT+F4** key combinations. You may also right-click the indicator to access commands to recompile the whole database, or to optimize the way in which device communications are organized. Manual database recompilation is rarely needed, as Crimson will typically perform the necessary steps without user intervention.

DOWNLOADING TO A DEVICE

Crimson database files are downloaded to the target device by means of the Link menu. The download process typically takes only a few seconds, but can take somewhat longer on the first download if Crimson has to update the firmware in the device, or if the device does not contain an older version of the current database. After this first download, Crimson uses a process known as incremental download to ensure that only changes to the database are transferred. This means that updates can be made in seconds, thereby reducing your development cycle time and simplifying the debugging process.

CONFIGURING THE LINK

The programming link between the PC and the target device can be made using an RS-232 port, a USB port or a TCP/IP connection. While TCP/IP connections are typically made via the panel's Ethernet port, they may also be established via a dial-in link. Before downloading, use the Link-Options command to ensure that you have the correct method selected…

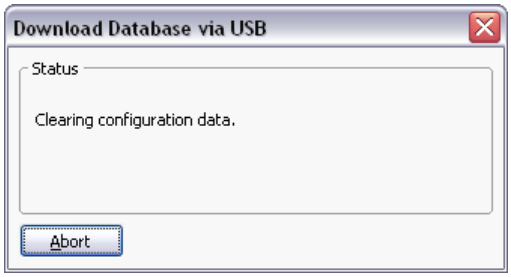


Note that this dialog does not provide any method to select the target IP address when using TCP/IP for download. This information is now stored in the database file and is configured via the Download tab of the Network configuration item. This method makes it easier to switch between multiple databases without having to re-configure the target IP every time.

Note also that Crimson maintain distinct download settings when working with multiple product families. This makes it easier to use USB for downloading to those products that support it, while falling back to serial download for less capable devices.

SENDING THE DATABASE

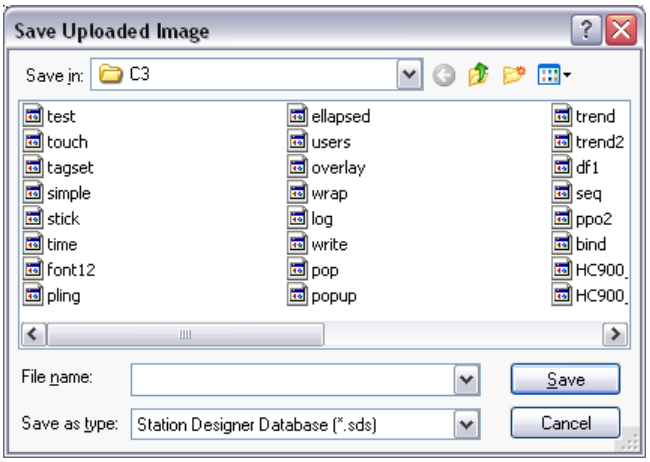
Once the link is configured, the database can be downloaded using either the Link-Send or Link-Update commands. The former will send the entire database, whether or not individual objects within the file have changed. The latter will only send changes, and will typically take a much shorter period of time to complete. The Update command is typically the only one that you will need, as Crimson will automatically fall back to a complete send if the incremental download fails for any reason. As a shortcut, you can access Link-Update via the lightning bolt symbol on the toolbar, or via the **F9** key on the keyboard.



Note that downloading via TCP/IP relies on a CompactFlash card being installed in the panel if the device's firmware is to be upgraded. Since you may want to perform such upgrades at some point in time, it is highly recommended that you install a CompactFlash card in any device to which TCP/IP downloads are likely to be performed. Note also that TCP/IP download must be enabled via the Network settings in the Communications category.

EXTRACTING DATABASES

The Link-Support Upload command can be used to instruct Crimson as to whether or not it should include the information necessary to support database upload when sending a database to a target device. This setting is stored in the database, and can thus be configured on a per-file basis. Supporting upload will slow the download process somewhat and may fail with extremely large databases containing many embedded images, but it will ensure that, should you lose your database file, you will be able to extract an editable image from the device.



Note that if you lose your database file and you do not have upload support enabled, you will not be able to reconstruct your file without starting from scratch. To extract a database from a panel, use the Link-Extract command. This command will upload the database, and prompt you for a name under which to save the file. The file will then be opened for editing. If the database was password-protected, you may be required to enter the password before if can be opened. In other words, enabling upload will not circumvent password protection.

SENDING THE TIME AND DATE

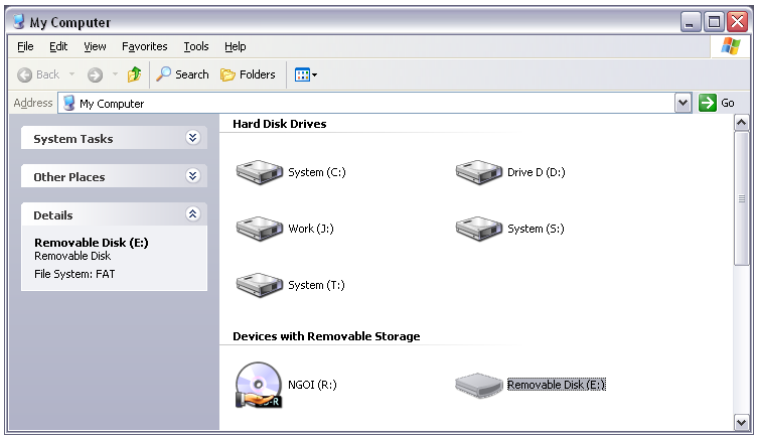
The Link-Send Time command can be used to set the target device's clock to match that of the PC on which Crimson is executing. This command also sends the current time zone and Daylight Savings Time settings to the target device, allowing the advanced features of the Time Manager to be used. Make sure your PC's clock is correct before you do this!

THE COMPACTFLASH CARD

If your target device has a CompactFlash card, several additional functions are available.

MOUNTING THE CARD

If you are connected to a suitable device via the USB port, you can instruct Crimson to mount the device's CompactFlash card as a drive within Windows Explorer. You can use this functionality to save files to the card or to read information from the Data Logger. The drive is mounted and dismounted by sending commands using the Mount Flash and Dismount Flash options on the Link menu. Once a command has been sent, the target device will be reset, and Windows will refresh the appropriate Explorer windows.



Note that some caution is required when mounting the CompactFlash card…

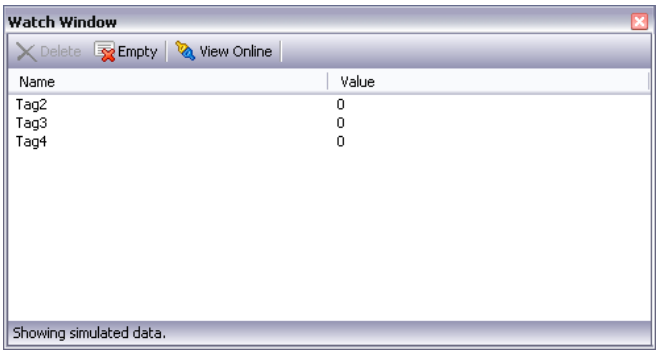
- When the card is mounted, the target device will periodically inform the PC if data on the card has been modified. This means that both the PC and the device will suffer a minor performance hit if the card is mounted during data logging operations for longer than necessary.
- If you write to the CompactFlash card from your PC, the target device will not be able to access the card until Windows releases its lock on the card. This may take several seconds, and will restrict data logging operations during that time, and prevent access to custom web pages. Crimson will use the device's RAM to ensure that no data is lost, but if too many writes are performed such that the card is kept locked for four minutes or more, data may discarded.
- You should never attempt to use Windows to format a CompactFlash card that you have mounted via Crimson, whether it be via Explorer or from the command prompt. Windows does not correctly lock the card during format operations, and the format may thus be unreliable and lead to subsequent data loss. See below for details of how to format a card in a reliable manner.

FORMATTING THE CARD

The only supported method of formatting a card is via the Format Flash command on the Link menu. Selecting this command will explain that the formatting process will destroy all the data stored on the CompactFlash card and offer you a chance to cancel the operation. If you elect to continue, the operator panel will be instructed to format the card. Note that this process may take several minutes for a large card. Slow formats on panels that are performing data logging may therefore result in gaps in the recorded data.

REMOTE MONITORING

Crimson supports as so-called Watch List that allows you to view the contents of the tags and mapping blocks contained within your database. The Watch List is displayed in the Watch Window. This can be shown or hidden using the **F7** key or the command on the View menu…



When first displayed, the Watch Window will show the simulated data that was defined when the tags were created. Pressing the View Online button will ensure that the current database matches that in the target device, and will then begin showing live data. The tag data will be displayed according to the appropriate format object.

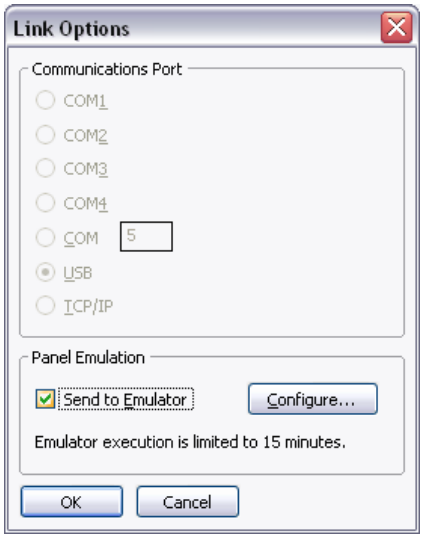
Items can be added to the Watch List by right-clicking and selecting the appropriate menu command. One or more tags can be added at once, as can the content of a mapping block. A command also exists to add all the tags references by a specific display page, thereby allowing easier debugging of the page you are working on. The buttons at the top of the Watch Window can be used to remove one item or all the item from the Watch List.

USING THE EMULATOR

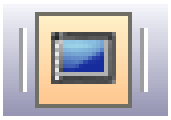
Crimson 3 contains an Emulator that is capable of executing configuration databases on the same PC that you are using for development purposes. The emulator is not just a simulation of the target device's user interface, but actually runs the same operating system and code, ensuring that you get an accurate representation of the target device's behavior.

ENABLING THE EMULATOR

The Emulator is enabled or disabled from the Link Options dialog box…



Its enable state may also be toggled using the Emulator button on the toolbar…



When the Emulator is enabled, the download process initiated from the Link menu or via the **F9** key will start the Emulator program, and then send the current database—or the database changes—to that process just as if it were talking to an actual target device. Pressing **F9** while running the Emulator will switch back to Crimson itself, making it easy to toggle between the two applications using the same key. Pressing **Esc** will close the Emulator.

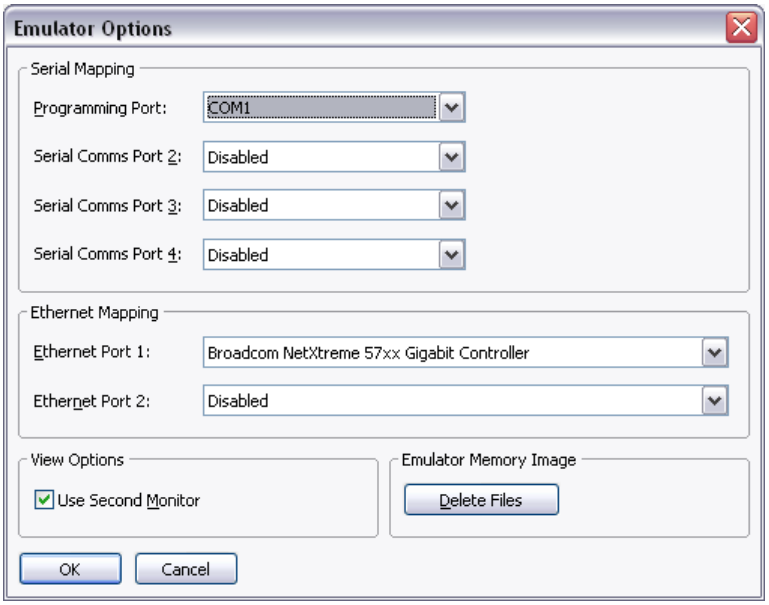
PORT MAPPING

The Emulator is capable of running any of the communications drivers provided by Crimson, and also implements the web server and other TCP/IP-based clients and services. In order for these facilities to work, Crimson must have access to the PC's serial or Ethernet ports. Each port on the target device can therefore be mapped to a given port on the host PC, such that, for example, COM1 might be used to represent the RS-232 Communications Port. If a port is unmapped, no comms activity will be performed for the associated devices.

Note that Ethernet ports are mapped such that the PC's port will appear to have a second MAC address in addition to its own. It will also have a further IP address allocated, either via DHCP or statically as per the Crimson database. The Emulator's addressing is thus distinct from that of your PC, and its web server and so on must therefore be referenced via the Crimson IP address rather than via that set in Windows.

EMULATOR CONFIGURATION

The Emulator configuration is accessed from the Link Options dialog…



The top section is used to define how the target device's serial port are mapped to the PC's ports, and the second section it used to similarly map the Ethernet ports. Not all ports will be used by every target device, as some may not have, say, dual Ethernet ports available.

The *Use Second Monitor* option can be used to force the Emulator to appear on a second display connected to your PC, making it easier to view the Crimson configuration and the resulting Emulator behavior at the same time.

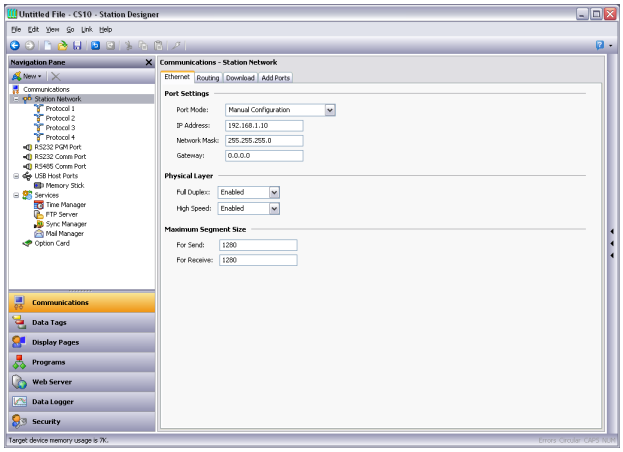
The *Delete Files* command can be used to clear the files that the Emulator uses to represent the various memory devices of the target hardware. This is not normally required, but can be used to resolve Emulator startup problems should they occur.

EMULATOR LIMITATIONS

Note that execution of the Emulator is limited to 15 minutes if any serial or network ports are mapped to physical hardware on the PC. With no ports mapped, execution time is unlimited.

USING COMMUNICATIONS

The first stage of creating a Crimson database is to configure the communications ports of the target device to indicate which protocols you want to use, and which remote devices you want to access. These operations are performed from the Communications category.



As can be seen, the Communications category lists the unit's available ports in the form of a tree structure. The example shown above has three primary serial ports, with the option to add a further two ports in the form of an expansion card. Target devices may also provide one or two Ethernet ports capable of running several communications protocols simultaneously.

SERIAL PORT SELECTION

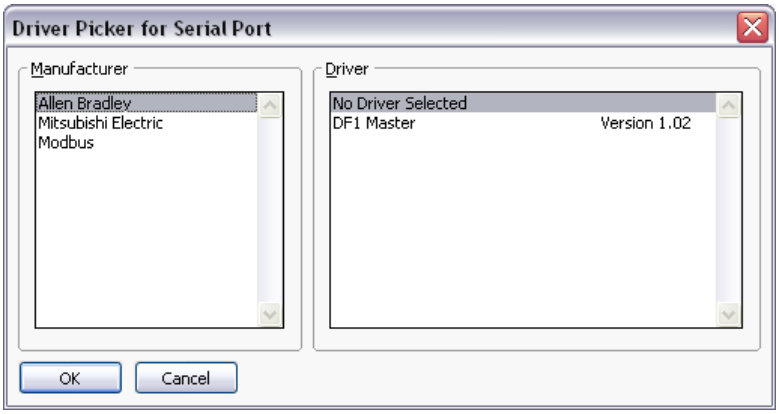
When deciding which of the target device's serial ports to use for communications, note that some devices (and certain option cards) multiplex a single serial controller between multiple ports. This implies that if either port is used for a slave protocol, the other port will be unavailable, and that if a token-passing protocol such as DH-485 is employed, the other port will similarly be disabled. Crimson will warn you if you attempt to create a configuration that breaks these rules.

Note also that a target device's programming port may be used as an extra communications port, but that it will not be available for download in these circumstances. This is not an issue if the unit has a USB port that is used for downloads, and it is therefore highly recommended that you use this method if you want to connect devices via the programming port. Where USB is not used, you must provide a method to re-enable serial downloads by executing the `StopSystem()` command in response to some user action.

SELECTING A PROTOCOL

To select a protocol for a particular port, click on that port's icon in the Navigation Pane, and press the Pick button next to the Driver field in the Editing Pane.

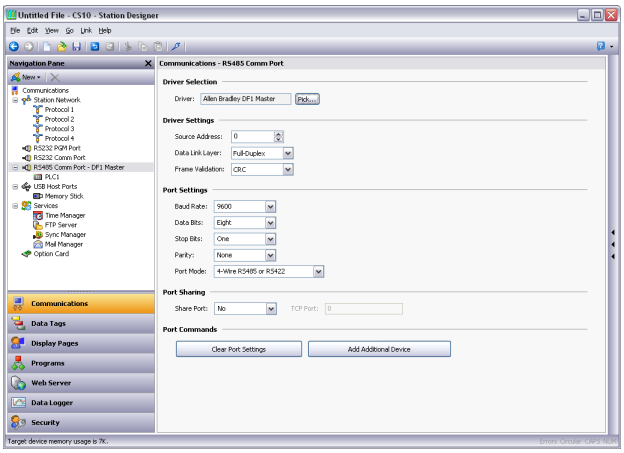
The following dialog box will appear…



Select the appropriate manufacturer and driver, and press the OK button to close the dialog box. The port will then be configured to use the appropriate protocol, and a single device icon will be created in the Navigation Pane. If you are configuring a serial port, the various Port Settings fields (Baud Rate, Data Bits, Stop Bits and Parity) will be set to default values appropriate to the protocol in question. You should obviously check these settings to make sure that they correspond to the settings for the device to be addressed.

PROTOCOL OPTIONS

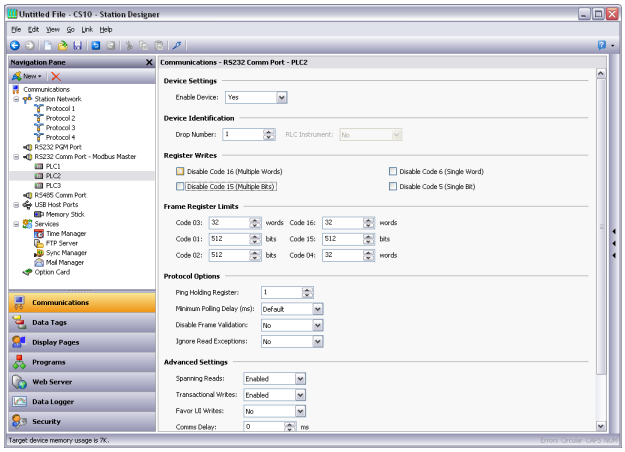
Some protocols require additional configuration of parameters specific to that protocol. These appear in the Editing Pane when the corresponding port icon is selected. The example below shows the additional parameters for the Allen-Bradley DF-1 driver, which appear under the Driver Settings section of the Editing Pane…



WORKING WITH DEVICES

As mentioned above, when a communications protocol is selected, a single device is created under the corresponding port icon. In the case of a master protocol, this represents the initial remote device to be addressed via the protocol. If the protocol supports access to more than one device, you can use the Add Additional Device button included in the Editing Pane to add further target devices. You may also use the New Comms Device command, accessed via the

Navigation Pane toolbar. Each device is represented via an icon in the Navigation Pane, and, depending on the protocol, may have a number of properties to be configured…



In the example above, the Modbus Universal Master protocol has been selected, and two additional devices have been created, indicating that a total of three remote devices are to be accessed. The Editing Pane shows the properties for each device. The Enable Device property is present for all devices, while the balance of the fields are specific to the protocol that has been selected. Note that the devices are given default names by Crimson when they are created. These names may be changed by selecting the appropriate icon in the Navigation Pane, pressing **F2** and then typing the new device name.

ADVANCED SETTINGS

In addition to the device settings mentioned above, certain master devices will also offer a number of advanced settings that can be used to optimize communications behavior…

- The *Spanning Reads* option specifies whether Crimson will optimize read operations by reading blocks of data, even if those blocks span registers that are not currently on the comms scan or referenced in the database. For example, with spanning reads enabled for a database that references registers D1, D2 and D4, a single comms command will be issued to read four registers from D1 onwards. Disabling spanning reads will results in two read operations, one for two registers from D1, and one for a single register from D4.

- The *Transactional Writes* options specifies whether a series of changes to a data value in Crimson should result in a corresponding series of write operations, or whether only the last written value be transferred. Transactional writes make, for example, pushbutton replacement easier.

- The *Favor UI Writes* option specifies whether to give priority to write operations that directly result from user actions. This is useful when working with a database that performs a lot of background communications as a result of protocol conversion or programmatic activity.

- The *Comms Delay* option specifies a delay that will be inserted between any two comms transactions for this device. It is useful when working with remote devices that are unable to keep up with Crimson's performance, or when a lower comms priority is to be given to a device.

CREATING TAGS

Some drivers provide an option to create tags within Crimson that correspond to the data items that exist in the remote device. This option is accessed via the Make Data Tags link on the device configuration page. The exact operation is driver-dependent, but typically you will be asked to select a configuration file that has been exported by the device's programming software. The import process will delete any tags from a previous import of the same device, but will preserve tag settings such as formats, triggers, security and so on.

PORT AND DEVICE USAGE

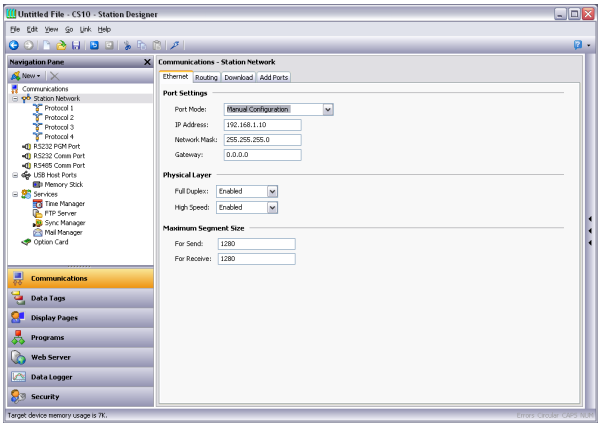
You can find all the items that refer to a given communication device or to any of the devices connected to a particular port by right-clicking that item in the Navigation Pane and selecting the Find Usage command. The resulting items will be placed on the Global Search Results List, and can be accessed by means of the **F4** and **SHIFT+F4** key combinations. The list itself can be shown or hidden by pressing **F8**.

NETWORK CONFIGURATION

The target device's IP network configuration is edited via the Network icon in the Navigation Pane. When the icon is selected, the Editing Pane will show a number of tabs, each of which allows a given set of properties to be configured.

ETHERNET SETTINGS

The first one or two tabs configure the target device's Ethernet ports…



PORT SETTINGS

The Port Mode field controls whether or not the port is enabled, and the method by which the port is to obtain its IP configuration. If DHCP mode is selected, the target device will attempt to obtain an IP address and associated parameters from a DHCP server on the network. If DHCP fails, an IP address will be allocated automatically using APIPA. (If the unit is configured to use slave protocols or to serve web pages, this option will only make sense if the DHCP server is configured to allocate a well-known IP address to the unit's MAC address, as otherwise, users will not be sure how to address the device!)

If the more common Manual Configuration mode is selected, the IP Address, Network Mask and Gateway fields must be filled out with the appropriate information. The default values provided for these fields will almost never be suitable for your application! Be sure to consult your network administrator when selecting appropriate values, and be sure to enter and download these values before connecting the target device to your network. If you do not do this, it is possible—although unlikely—that you will cause problems on your network.

Selecting IEEE 802.3 Only mode will enable the port for low-level communications, but will not allocate an IP address or allow TCP or UDP to operate. This mode only makes sense when using drivers that use raw Ethernet, such as certain building automation protocols.

PHYSICAL LAYER

The Physical Layer options control the type of connection that the device will attempt to negotiate with the hub or switch to which it is connected. Generally, these options can be left in their default states, but if you have trouble establishing a reliable connection, especially when connecting directly to a PC without an intervening hub or switch, consider turning off both Full Duplex and High Speed operation to see if this solves the problem.

MAXIMUM SEGMENT SIZE

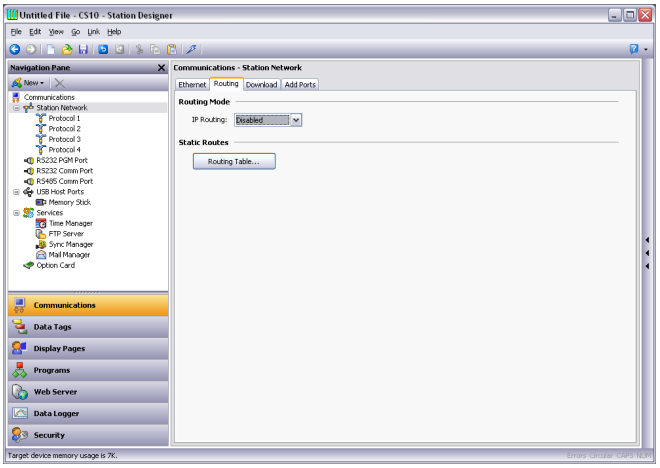
The Maximum Segment Size options control the MSS settings for TCP send and receive. You should not generally have to change these settings as the default values are suitable for virtually all applications and all networks.

MULTIPLE PORTS

If you are using more than one Ethernet port, note that only a single port should have a default gateway defined, and that each port should have a distinct network address. Crimson will only ever send a given IP packet to a single interface, so a configuration that, for example, defines the first Ethernet port as 192.168.100.1 and the second as 192.168.100.2 will result in all packets for the 192.168.100.0 network going to the first port, thereby preventing the second port from operating correctly.

ROUTING SETTINGS

The second tab configures Ethernet routing options…

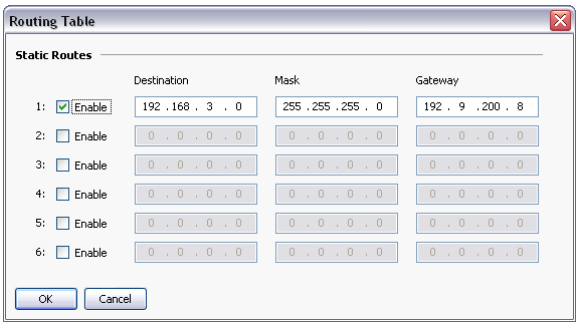


ROUTING MODE

The IP Routing option is used to enable or disable packet routing between interfaces. If this option is enabled, IP packets received on an Ethernet or modem port that are destined for devices connected to another port will be forwarded as required. Disabling this option will prevent such forwarding. The required setting will be dependent on your network topology.

ROUTING TABLE

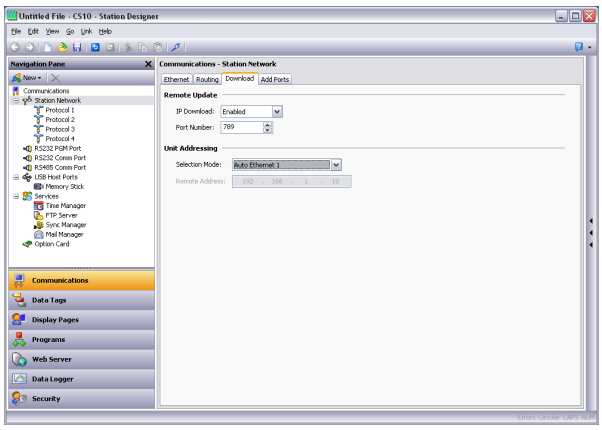
The routing table defines additional static routes for Crimson's TCP/IP stack.



In the example above, a single route has been specified, telling Crimson to forward any packets destined for IP addresses starting with 192.168.3 to the router located on the local network at address 192.9.200.8. Once again, the exact settings required will be dependent on the topology of the network to which the target device is connected.

DOWNLOAD SETTINGS

The third tab is used to configure downloads to the target device over TCP/IP…



REMOTE UPDATE

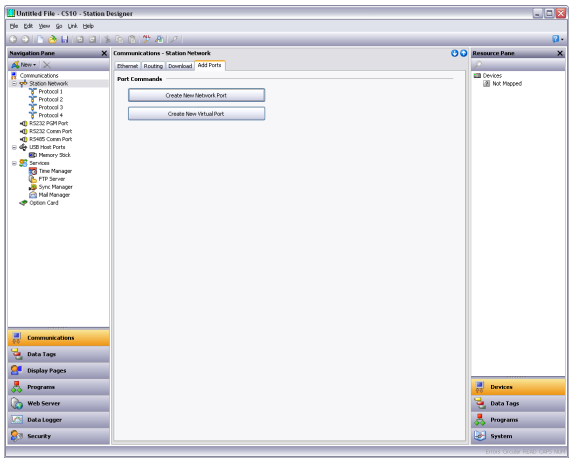
The IP Download option is used to enable or disable TCP/IP downloads, while the Port Number option specifies which TCP port should be used for such downloads. The default value of 789 should be used unless you have a good reason to use something else.

UNIT ADDRESSING

These settings are used to specify the IP address to be used by the Crimson configuration software when the TCP download method is selected in the Link-Options dialog box. Auto mode will use the IP address configured for the selected Ethernet port. (Obviously, the port must be manually configured for this to make sense.) Manual mode allows an IP address to be entered via the Remote Address field. Note that this information is saved as part of the database, allowing you to easily switch between units on the same network.

ADDING PORTS

The fourth tab can be used to add additional network protocols…



Pressing the Create New Network Port button will add a further network protocol, up to the maximum number of ports supported by the target device. Pressing the Create New Virtual Port button perform a similar operation, but will add a port capable of emulating a serial connection over TCP/IP. Either type of port can be deleted by selecting it in the Navigation Pane and by pressing **ALT+DEL** or by selecting the delete toolbar option.

PROTOCOL SELECTION

Once the network has been configured, you can select the protocols that you wish to use for communications. Several protocols may be used at once, and many of these protocols will support multiple remote devices. This means that you have several options when deciding how to mix protocols and devices to achieve the results you want.

For example, suppose you want to connect to two remote slave devices using Modbus over TCP/IP. Your first option is to use two network protocols, configuring both as Modbus masters with a single device attached to each. For most protocols, this will produce higher performance, as it will allow simultaneous communications with the two devices. It will, however, consume two of the available protocols, limiting your ability to connect via additional protocols in complex applications.

Your second option is therefore to use a single protocol configured as a Modbus TCP/IP Master, but to add a further device so that both slaves are accessed via the same driver. This will typically produce slightly reduced performance, as Crimson will poll each device in turn, rather than talking to both devices at the same time. It will, however, conserve network protocols, allowing more complex applications without running out of resources.

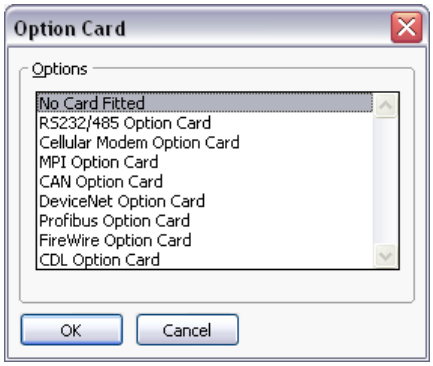
USING VIRTUAL PORTS

As mentioned above, Crimson supports the addition of virtual ports to the network configuration. A virtual port looks to Crimson's communications system just like a serial port, but sends and receives its data over a TCP/IP link. Virtual ports may be configured in either active mode or passive mode. In the former case, Crimson will attempt to open a TCP/IP connection to a specified remote device, while in the latter case, Crimson will listen on a specific TCP/IP port for incoming connections. Virtual ports are typically used to communicate with devices via remote serial servers: A standard serial protocol is employed, but that protocol's data is being encapsulated within TCP/IP packets.

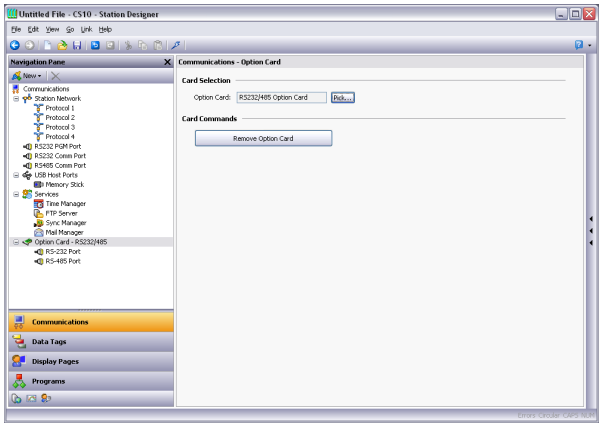
USING EXPANSION CARDS

Some target devices support the addition of one or more expansion cards so as to provide additional communications facilities. A number of cards are available, including models to support bus protocols such as CANOpen, Profibus or DeviceNet. Installation instructions are provided with each card, so please refer to the supplied data sheet for information on how to fit the card to the device. Once the card is installed, configuration is performed by selecting the appropriate icon in the Navigation Pane, and clicking on the Pick button next to the Option Card property.

A dialog similar to the one shown below will be displayed…



Selecting the appropriate card will add one or more icons to Navigation Pane, representing the additional port or ports that are made available by the card. These ports can then be configured in the usual way. The example below shows a serial expansion card …



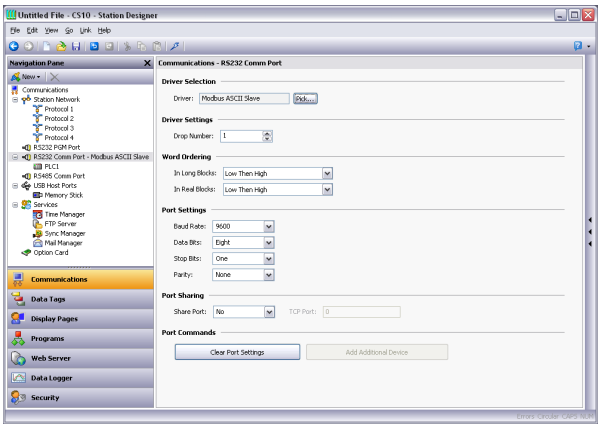
The additional ports can be configured by following the instructions supplied in the previous sections. Note that the drivers available for a port will depend on the connection type it supports. For example, the CANOpen expansion card shows a port that will only support drivers designed for the CAN communication standard.

SLAVE PROTOCOLS

For master protocols (i.e. those where the Crimson device initiates communication) there is no further configuration required under the Communications category. For slave protocols (i.e. those where the Crimson device receives and responds to remote requests), the process is slightly more complex, as you must also indicate what data you wish to expose.

SELECTING THE PROTOCOL

As with master protocols, the first stage is to select the protocol for the communications port that you wish to use. The example below shows the target device's RS-232 port configured for operation with the Modbus ASCII Slave protocol…

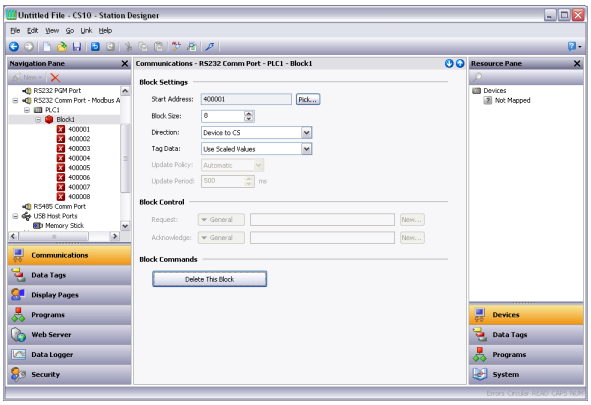


Note that a single device has been created for the protocol. In the case of master protocols, this represents the remote device that Crimson will access. In this case, though, the device represents the Modbus slave that the hardware will itself embody. This means that only a single device is required, and that things such as the station number to which the hardware will respond are normally configured via the port settings rather than those of the device.

ADDING GATEWAY BLOCKS

Having configured the protocol, you must now decide what range of addresses you want the slave protocol to expose. In this example, we want to use Modbus registers `40001` through `40008` to allow read and write access to certain data items in our database. We begin by selecting the device icon in the Navigation Pane, and clicking the Add Gateway Block button in the Editing Pane.

An icon representing the block will appear under the device…

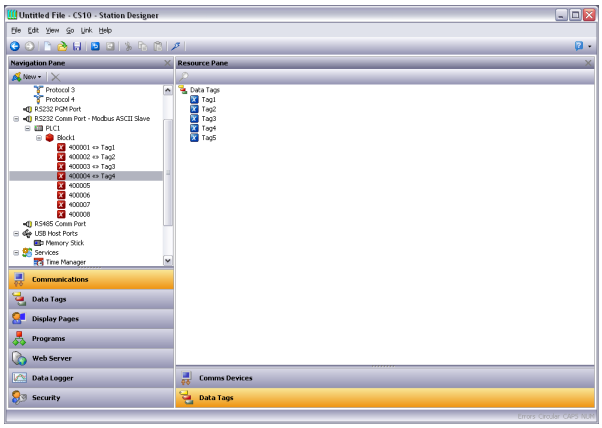


In the example above, we have configured the Start Address to `40001` to indicate that this is where we want the block to begin. We have also configured the Block Size to eight so as to allocate one Modbus register for each tag we want to expose, and we have configured the

Direction as Device to Crimson, to indicate that we want remote devices to be able to read and write data items exposed via this block. Finally, we have left the Tag Data property at its default setting of Use Scaled Values, indicating that we want any scaling to be applied to tag data before that data is transferred to the gateway block.

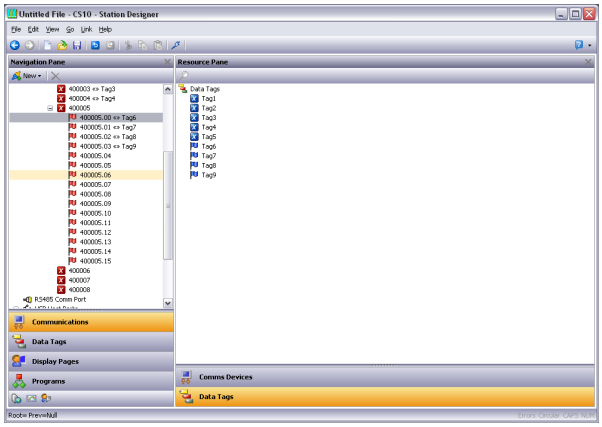
ADDING ITEMS TO A BLOCK

Once the block has been created and its size defined, entries appear in the Navigation Pane to represent each of the registers that the block exposes to remote access. When one of these entries is selected, an expanded Resource Pane appears and provides access to available data items. These items comprise both tags from within your database, and data registers from any master communications devices that you have configured…



To indicate that you want a particular register within your gateway block to correspond to a particular data item, simply drag that item from the Resource Pane to the Navigation Pane, dropping it on the appropriate gateway block entry. The example above shows how the first four registers in the block have been mapped to tags called `Tag1` through `Tag4`, indicating that accesses to `40001` through `40004` should be mapped to the respective variables.

ACCESSING INDIVIDUAL BITS



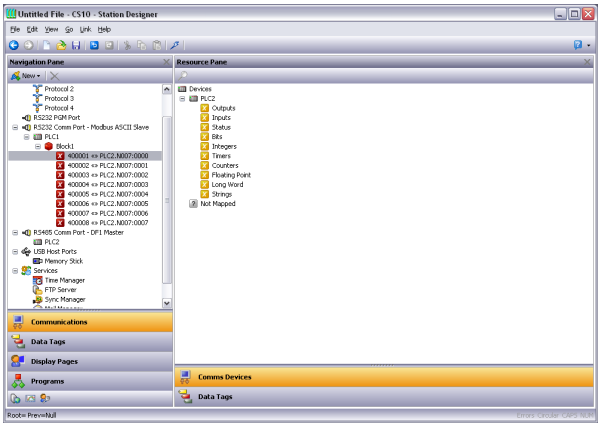
If your application requires it, you can expand individual elements within a Gateway Block to their constituent bits, and map a different data item to each bit. To do this, right-click on the element in question, and select Expand Bits from the resulting menu. The Navigation Pane

will be updated to show the individual bits that make up the register, and these can be mapped using the drag-and-drop process described above.

PROTOCOL CONVERSION

In addition to exposing internal data tags via slave protocols, Gateway Blocks can also be used to expose data that is obtained from other remote devices, or to move data between two such master devices. This unique protocol conversion feature allows much tighter integration between elements of your control system, even when using simple, low-cost devices.

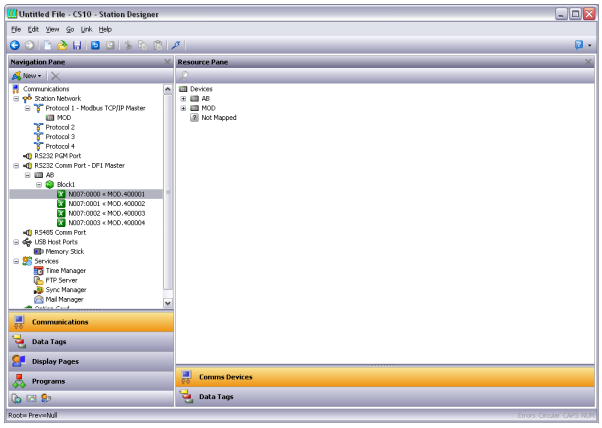
MASTER AND SLAVE



Exposing data from other devices over a slave protocol is simply an extension of the mapping process described above, except this time, instead of dragging a tag from the Resource Pane, you should select the Comms Devices category, expand the appropriate master device, and drag across the icon that represents the registers that you want to expose. You will then be asked for a start address in the master device, and the number of registers to map, and the mappings will be created as shown.

In this example, registers `N7:0` through `N7:7` in an Allen-Bradley controller have been exposed for access via Modbus TCP/IP as registers `40001` through `40008`. Crimson will automatically ensure that these data items are read from the Allen-Bradley PLC so as to fulfill Modbus requests, and will automatically convert writes to the Modbus registers into writes to the PLC. This mechanism allows even simple PLCs to be connected on an Ethernet network.

MASTER AND MASTER



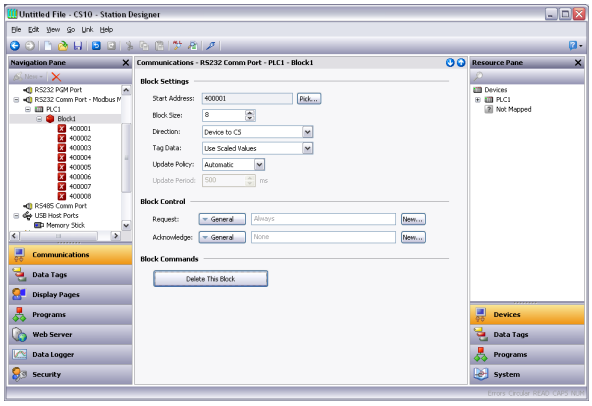
To move data between two master devices, simply select one of the devices, and create a Gateway Block for that device. You can then add references to the other device's registers just as you would when exposing data on a slave protocol. Again, Crimson will automatically read or write the data as required, transparently moving data between the devices. The example above shows how to move data from a Modbus device into an SLC-500.

WHICH WAY AROUND?

One question that may occur to you is whether you should create the Gateway Block within the Allen-Bradley device, as in this example, or within the Modbus device. The first thing to note is that there is no need to create more than a single block to perform transfers in a single direction. If you create a block in AB to read from MOD, and a block in MOD to write to AB, you'll simply perform the transfer twice and slow everything down! The second observation is that the decision as to which device should own the Gateway Block is essentially arbitrary. In general, you should create your blocks so as to minimize the number of blocks in the database. This means that if the registers in the Allen-Bradley lay within a single range, but the registers in the Modbus device are scattered all over the PLC, the Gateway Block should be created within the Allen-Bradley device so as to remove the need to create multiple blocks to access the different ranges of the Modbus address space.

CONTROLLING MASTER BLOCKS

Gateway blocks within master devices have several additional properties …



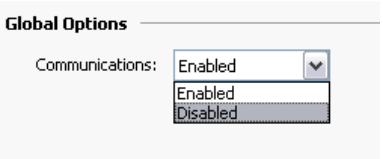
- As with slave blocks, the *Tag Data* property selects how data tags are mapped to and from the block. As you will discover in the next chapter, a tag data can be subject to various stages of transformation. This property selects where in the transformation process the gateway block will obtain and inject its data.
- The *Update Policy* property is used to define how the block updates. The default setting of Automatic will cause read blocks to update continuously, and write blocks to transfer only those values that have changed. A setting of Continuous will cause all blocks to update continuously. A setting of Timed will cause all blocks to update at the rate defined by the *Update Period* property, with the entire contents of a write block being written each time.
- The *Request* and *Acknowledge* properties are used to control the timing of block updates via tags or other data items. If the Acknowledge property is left empty, Request will act as an enable field, with a zero value disabling the block and a non-zero value allowing it to operate. If the Acknowledge is defined, the Request and Acknowledge will operate as a standard two-wire handshake, with the block updating once on each rising edge of the Request, and the Acknowledge being set after the transaction completes.

DATA TRANSFORMATION

You may also use Gateway Blocks to perform math operations that your PLC might not otherwise be able to handle. For example, you may want to read a register from the PLC, scale it, take the square root, and write it back to another PLC register. To accomplish this, refer to the section on Data Tags, and create a mapped tag to represent the input value that will be read from the device. Then, create a tag to represent the output value, setting the expression so as to perform the required math. You can then create a Gateway Block targeted at the required output register, and drag the formula across to instruct Crimson to write the derived value back to the PLC.

DISABLING COMMUNICATIONS

Crimson provides the option to disable all driver-based communications by means of a property contained in the top-level item of the Communication category…



Disabling communications can be useful during development when you do not have the remote devices available at your current location. When operating in disabled mode, Crimson initially sets all tags equal to their simulated values, and then allows them to be changed just as if they were being written to the associated devices. If you find your communications has stopped for no reason, make sure you do not have this setting set to disabled!

WORKING WITH TAGS

Once you have configured the communications options for your database, the next step is to define the data items that you want to display or otherwise manipulate. This is done by selecting the Data Tags category in the Navigation Pane. Tags can be created, deleted and otherwise manipulated using the standard operations referred to earlier in this manual.

ALL ABOUT TAGS

Data Tags are named entities that represent data items.

DATA SOURCES

Tags may get their data from three possible sources…

- A tag may be *mapped* to one or more registers in a remote device, in which case Crimson will automatically read the corresponding register when the tag is referenced or displayed. Similarly, if you change a mapped tag, Crimson will automatically write the new value to the device.
- A tag may be *internal*, in which case it represents one or more data elements within the Crimson-based device. Internal tags can be marked as retentive, in which case they will keep their values through a power-cycle, or non-retentive, in which case they will be reset to zero on power-up.
- A tag may be an *expression*, in which case it represents a calculation based upon other data items, optionally using mathematical operators and one or more of Crimson's internal or user-defined functions. Expression tags are used to calculate derived values for internal processing or for transfer to remote devices.

TYPES OF TAGS

Crimson supports three main types of tags…

- *Numeric Tags* represent integer or floating point values.
- *Flag Tags* represent an on-or-off value.
- *String Tags* represent strings of Unicode characters.

Each of the three main tag types can represent a single value or an array of values. An array is a collection of items with similar properties that are grouped together and accessed via an index value. Mapped arrays correspond to multiple registers in the target device.

A fourth type of tag is the *Basic Tag*. This is a simplified version of a tag that can only represent string or numeric expressions. It lacks many of the powerful features of the standard tags. It is typically used to encode simple data items like constants.

TAG ATTRIBUTES

Tags within Crimson are rich objects that define various common properties…

- A tag's *label* is a translatable, human-readable string used to automatically label data fields referring to this data item. It is also used by the Web Server and the Data Logger to label associated data items.
- A tag's *description* is an non-translatable string used to provide an annotation as to the tag's purpose. It is not normally viewed by the user of the target device, but can be displayed for diagnostic purposes.
- A tag's *format* is a collection of settings that define the method by which the tag data is to be presented for display. The format may be left as General, in which case Crimson will use default formatting rules, or may be set to one of many formatting types. For example, a numeric value may be displayed in scientific format, or may be used to select a number of different text strings.
- A tag's *coloring* is a collection of settings that define how the tag's text is to be displayed or what colors are to be used to represent the state of the tag. Again, a number of different colorings exist, allowing a tag to change its appearance based upon a variety of conditions. Foreground and background colors are defined in pairs, and can be accessed individually by display primitives.
- A tag's *security descriptor* defines the access rules to be used when changing the tag, and whether or not those changes are to be logged.

Basic tags lack format, coloring and security information. In addition, numeric and flag tags define alarms and triggers, respectively allowing alarms to be fired or actions to be taken by the occurrence of certain conditions.

ADVANTAGES OF TAGS

Since Crimson allows you to place a PLC register directly on a display page without going to the trouble of defining data tags, it is worth spending a moment pointing out the benefits of the minimal extra work that is involved with using tags …

- Tags allow you to name data items, so you know which data item within the PLC you are referring to. Further, if the data in the PLC moves or if you decide to switch to an entirely different family of PLC, you can simply re-map the tags, and avoid having to make any other changes to your database.
- Tags allow you to avoid re-entering the same information again and again. When you create a tag, you specify how the tag is to be displayed. In the case of a numeric tag, this means you tell Crimson how many decimal places are to be used, and what units, if any, are to be appended to the value. When you place a tag on a display page, Crimson knows how to format it without you having to do anything further. Similarly, if you decide to change the formatting, and perhaps switch from one set of units to another, you can do this in one place, without having to edit each display page in turn.

- Tags are used as one basic method for color animation. The various colors that are defined for a tag can be used to specify the way in which other animation primitives will be displayed. While there are other methods, tags provide a simple way of changing the color of display primitives.
- Tags are the key to implementing slave protocols. Crimson treats these protocols as mechanisms for exposing data items within the terminal. This allows the same data to be accessed via multiple ports, so that, for example, a machine setting could be changed by both a local SCADA package, and a similar package working over Ethernet from a remote site. Without tags, there would be nothing to expose, and this mechanism could not be implemented!
- Tags are used within Crimson to implement many advanced features. If you want to use functionality such as alarms, triggers, data logging or the web server, you will have to use tags, period. The formatting data from the tag definition is typically required by all these features, so tags are mandatory for their operation.

  In other words, tags will automate many tasks during programming, saving you time. Even if you decide not to use tags, many of the subsequent chapters of this manual refer to concepts discussed in this chapter. You should thus read them thoroughly before proceeding.

EDITING PROPERTIES

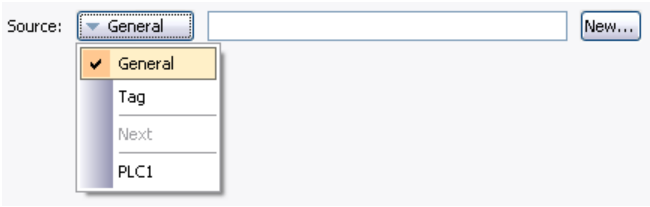
Most properties are edited in ways that are self-evident to anyone who has used a Windows operating system. For example, you may be required to enter a numeric value, or to select an item from a drop-down list. Certain types of property, though, provide more complex editing options, and these are described below.

EXPRESSION PROPERTIES

Expression properties are capable of being set to…

- A constant value.
- The contents of a data tag.
- The contents of a register in a remote communications device.
- A combination of such items linked together using various math operators.
- The return value of a local program.

In its default state, the arrowed button immediately after the label of the property shows that the field is in General mode. The edit box to the right of the button may show a grayed-out string that indicates the default behavior of the property. An example of an empty expression property without a default value is shown below…



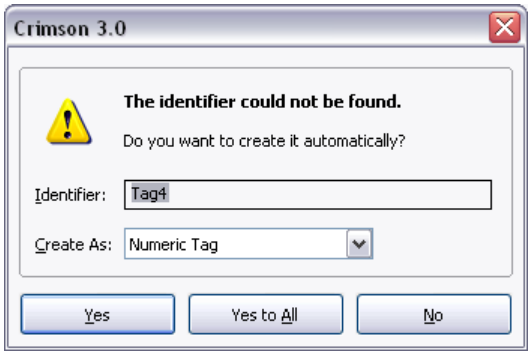
If you are familiar with Crimson's expression syntax—a complete description of which can be found in the Writing Expressions chapter—you can edit the property by typing an expression directly into the edit box.

SELECTING A TAG

To set an expression property to an existing tag, you have four options. First, you can ensure that the target field is selected and then double-click the required tag in the Resource Pane. Second, you can drag the tag from the Resource Pane and drop it on the target field. Third, you can select Tag from the drop-down menu activated by the arrowed button and be reminded that you could just have dragged the target to the field in the first place! Finally, you can just do it the old fashioned way and type the tag name into the expression property.

CREATING A TAG

To set an expression property to a new tag, you have three options. First, for expressions that define the source of a data item, you can select the New Tag option in the drop-down menu activated by the arrowed button. Second, if you already have a tag selected, you can press the Pick button and select New Tag from the resulting dialog box. Finally, you can enter the name of the new tag as part of an expression, and have Crimson prompt you via a dialog similar to that shown below…



In this example, an expression referencing Tag4 has been entered, but no such tag exists within the database. Crimson spots the error, and asks if you would like to create this tag automatically. The drop-down list can be used to select the type of the new tag, and will contain options appropriate for the context in which the tag was used. The Yes to All button

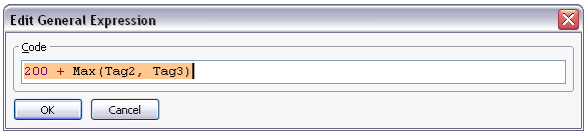
can be used to tell Crimson to use the default data type to create any other missing tags contained within this expression without any further prompting.

COMMS REFERENCES

To select a register from a comms device, select a device from the drop-down menu. A dialog box will be displayed, allowing you to choose a register within that remote communications device. The various communications devices are listed at the end of the menu in the order in which they were created. You may also select the Next option from the drop-down menu, thereby setting the current tag equal to the last-used PLC register plus the number of registers mapped to that address. For example, mapping a 32-bit tag to Modbus register `40001` and then selecting Next will map the subsequent tag to `40003`.

EDITING AN EXPRESSION

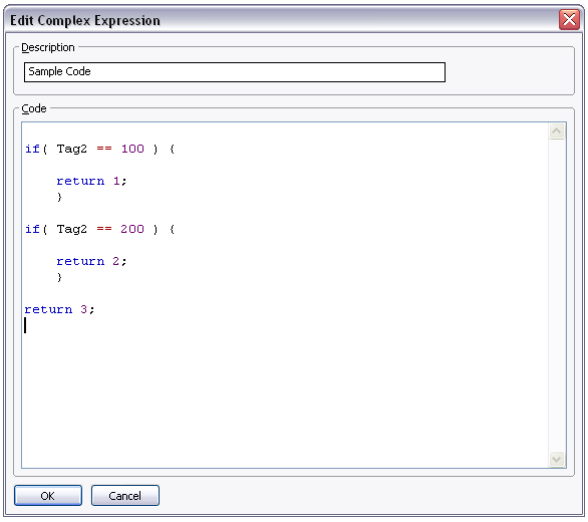
As mentioned above, general expressions are typically edited directly in the edit box of the property. However, they can also be edited by pressing the Edit button next to the field, thereby activating a dedicated dialog box that allows more of the expression to be seen…



The editor used in this dialog box is the same as is used for creating global programs. It therefore provides syntax coloring. You can also access help information on system functions by placing your cursor in or at the end of the function name and pressing **F1**.

COMPLEX EXPRESSIONS

If your expression is too complex to fit into a single line—or if you would simply rather write it as a program—you may select the Complex option from the drop-down menu, thereby allowing a local program to be created…



The `return` statement is used provide the value of the expression, just as if you had called a global program. Note once again that the program editor is used, providing syntax coloring and auto-indent facilities, and that the `F1` mechanism described above can be used to consult help information on system functions. The Description text allows you to make a quick note of the program's function. This will be displayed next to the property for reference. For information on writing and editing programs, refer to later chapters in this manual.

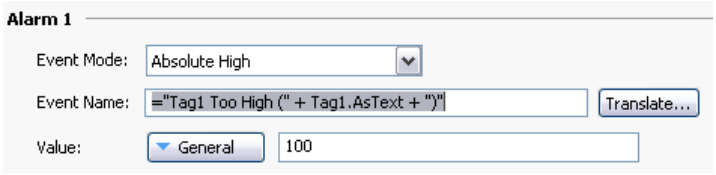
TRANSLATABLE STRINGS

Crimson databases are designed to support multilingual operation, whereby any string that will be presented to the user of the target device is capable of being displayed in one of many different languages. To allow you to define these translations, properties that contain such strings have a button labeled Translate to their right-hand side.

To enter the translations, click the button and the following dialog box will appear…

The languages listed in the dialog are defined at the database level. Refer to the chapter on Localization for information on how they are selected, on the operation of the Auto-Translate function, and on how to switch the language at runtime. Note that if you do not enter text for a particular language, and that language is subsequently selected by the operator, Crimson will use the text from the default language instead.

Translatable strings are also capable of being defined as expressions, thereby allowing them to change at runtime. For example, while an alarm name would typically be set during configuration, a database designer might want the alarm to contain the value of the tag that triggered the alarm. Expressions can be entered by prefixing them with an equals sign, just as one would do when editing a spreadsheet, as shown in the example below…



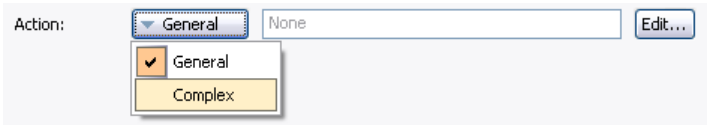
Note the use of the `AsText` property of the tag to allow its value to be accessed as a string according to its format setting. Refer to the Writing Expressions chapter for more details.

TWO-WAY PROPERTIES

Properties such as translatable strings that are capable of being set to a constant value or an expression are call two-way properties. As well as accepting expressions prefixed with an equals sign, they can be set to tag values by simply dragging the appropriate tag from the Resource Pane and dropping it on the field.

ACTION PROPERTIES

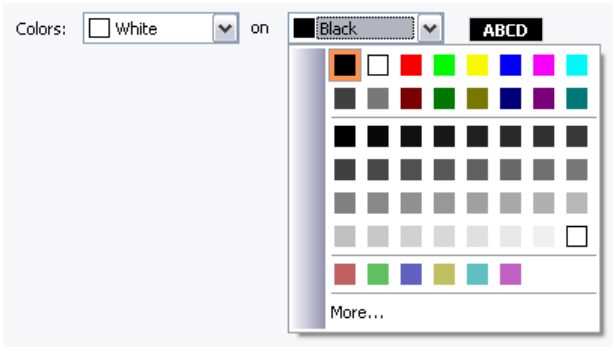
Action properties are used within tags to define action to be performed when triggered conditions are met, or when a tag value is changed. They are edited by a drop-down and edit box similar to those used for editing expressions…



As with expressions, the Edit button can be used to invoke a larger editing window, and complex actions can be created by means of local programs.

COLOR PROPERTIES

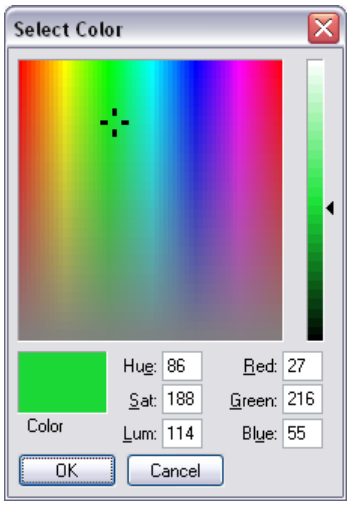
Color properties within tags represent a pair of colors, these being the foreground and a background color that may be used when displaying the tag's state in textual form. The example below shows a color pair being edited…



The drop-down menu contains the following colors…

- The sixteen standard VGA colors.
- Thirty-two shades of gray between black and white.
- Any other colors used in the database, up to a limit of twenty-four.

The More option at the bottom of the list can be used to invoke the color selection dialog…



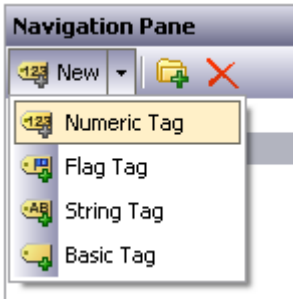
This dialog offers several ways of defining a color. You can pick from the palette, pick from the "rainbow" window, or enter the explicit HSL or RGB parameters. If the color selected has not previously been used in the database and is not one of the standard colors or grays, it will be added to the custom colors shown in the drop-down menu.

LOG PROPERTIES

When you first enter the Data Tags category of the Navigation Pane, you will notice a number of properties relating to event logging. These properties control if and how events generated by tags or by their alarms will be saved to CompactFlash. They are analogous to the properties defined by data logs, and you are thus referred to the Using the Data Logger chapter later in this manual for more information on how they can be used.

CREATING TAGS

Data tags are created and otherwise manipulated via the usual methods in the Navigation Pane. You will notice that you can create folders to organize your tags, and that the New button in the toolbar contains a drop-down arrow to allow you to select the type of tag to be inserted. The left-hand side of the New button will create tag of the same type as the last one you created, making it easier to create multiple tags without using the drop-down list.



DUPLICATING TAGS

The Smart Duplicate command on the Edit menu can be used to create a new copy of an existing tag, incrementing its data source to refer to the next data element.

The definition of "next" depends on the exact type of the data element, with Crimson being capable of selecting the next register in a comms device, the next member of an array, or the next tag in a sequence. For example, using Smart Duplicate on a 16-bit tag mapped to Modbus register `40001` will produce a tag mapped to `40002`, while using it on a tag mapped to `Array[2]` will produce a tag mapped to `Array[3]`.

This facility makes it much easier to create sets of tags referring to sequential data items.

EDITING MULTIPLE TAGS

You may on occasions want to edit the properties of several tags at once. Crimson supports this operation by having you edit one tag, and then allowing you to set the properties of the other tags equal to those of the one that you first edited. Crimson provides two methods to do this, both of which rely on the same underlying mechanism.

USING COPY FROM

The Copy From command can be used to copy the selected properties of a given tag to one or more tags in the Navigation List. To use the command, select the target tags, and then right-click to access the context menu. (Note that the Navigation List for tags supports multiple selection via the usual **SHIFT** and **CTRL** key combinations.) Select one of the Copy From commands, and the cursor will change to allow you to select the tag from which the copy operation should be performed. Depending on the command that was selected, one or more properties from the source tag will then be applied to the target tags.

USING PASTE SPECIAL

The Paste Special command can be used to achieve the same result, but via a different method that also allows properties to be copied between databases and or between multiple instances of Crimson. First, select the source tag and use the Copy command to put it and its properties on the Clipboard. Next, select the target tags in the Navigation List, again noting that multiple selections can be made. Finally, right-click the selection to access the context menu, and select the Paste Special command. The following dialog box will appear…

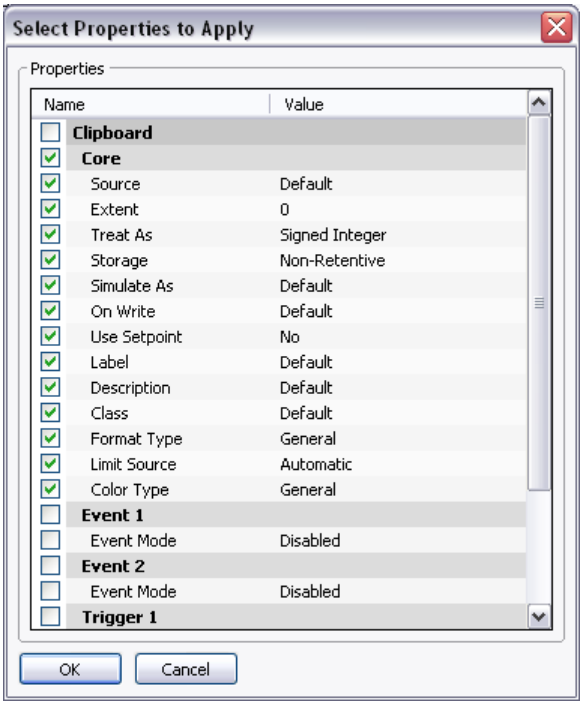
The selected properties from the source tag will be applied to the target tags.

PROPERTY SELECTIONS

Both methods detailed above allow you to define which properties are to be copied…

- *Mapping* copies the Source property. It also copies all the properties that control that tag's communications options, such as the Extent, Access, and all the other properties that are contained in the Data Source section.
- *Scaling* copies the Scale To property and the associated scaling limits.
- *Format* copies the Format Type, and the associated Format object.
- *Coloring* copies the Coloring Type, and the associated Coloring object.
- *Alarms* copies all the properties of Alarm 1 and Alarm 2.
- *Triggers* copies all the properties of Trigger 1 and Trigger 2.
- *Security* copies all the properties from the tag's Security page.

In addition, the *Selective* option can be used to select the properties to copy…



The list contains a hierarchical presentation of all the properties defined by the source tag, organizing them according to the layout used when editing the tag, and showing the value assigned to each. The properties or groups of properties can be selected or deselected using the associated checkboxes. Only the checked properties will be applied, providing you with low-level control of what gets copied from one tag to another.

IMPORTING AND EXPORTING

Selecting the Data Tags item in the Navigation List provides access to buttons that can be used to export and import the data tags in your database. Tags may be exported to either Unicode text file, or ANSI comma separate variable files, with either capable of being edited via applications such as Microsoft Excel. The export file is divided into sections based upon tag type, format type and coloring type. Each section contains a number of columns, the meanings of which can be determined by consulting the sections below.

Note that certain communications drivers have the ability to import, for example, a PLC configuration file and create data tags that correspond to the PLC registers. This facility is accessed via the device configuration page using the Make Data Tags command.

FINDING TAG USAGE

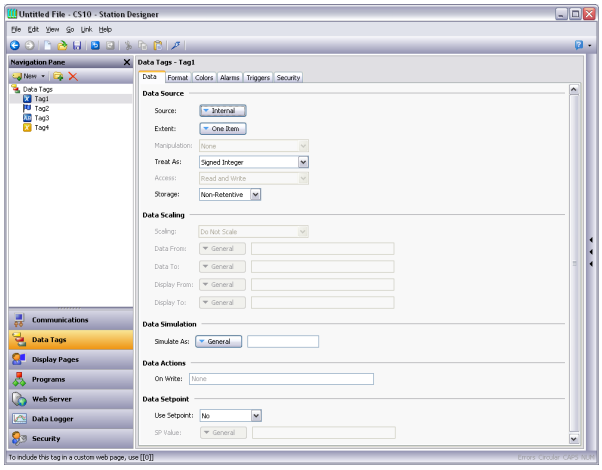
You can find all the items that refer to a given tag by right-clicking that item in the Navigation Pane and selecting the Find Usage command. The resulting items will be placed on the Global Search Results List, and can be accessed by means of the **F4** and **SHIFT+F4** key combinations. The list itself can be shown or hidden by pressing **F8**.

NUMERIC TAGS

A numeric tag represents one or more integer or floating point values. Crimson performs all internal calculations using either 32-bit signed integers or single-precision floating point, so all data will be converted to one of these forms before processing. Mapped numeric tags support a number of transformations that occur between the raw data and the data that will be used by Crimson. The exact process is defined in detail later in this chapter.

DATA PROPERTIES

A numeric tag has the following properties on its Data tab…



DATA SOURCE

- The *Source* property defines where the tag gets its data. The default setting results in an internal tag, but the drop-down list may be used to select a general expression, another data tag or an item from a remote device.
- The *Extent* property is used choose between a single element tag or an array. If you select an array, you must enter the required number of elements. Arrays are not permitted for tags whose source is an expression. For mapped items, the number of registers to be read from the remote device depends upon the data type defined for the address. For example, an array of two elements that was mapped to a register of type Word as Long will result in four registers being accessed, with two words being needed for each long value. A similar array mapped to a data type of Word as Word will only need two registers.
- The *Manipulation* property defines the first-stage transformation that is applied as comms data is being transferred into a mapped tag. The following options may be available, depending on the exact data type being used…

MANIPULATION	DESCRIPTION
None	The data will not be changed.
Invert Bits	Each bit in the data will have its state inverted.

MANIPULATION	DESCRIPTION
Reverse Bits	The most significant bit in the data will be swapped with the least significant bit, with intermediate bits being treated in a similar fashion.
Reverse Bytes	The most significant byte in the data will be swapped with the least significant byte and so on. Only available for data items of 16 bits or more in size.
Reverse Words	The most significant word in the data will be swapped with the least significant word. Only available for data items of exactly 32 bits in size.
BCD to Binary	Each four bit group in the data will be interpreted as a single decimal digit. Selecting this option will force the data to an unsigned integer.

- The *Treat As* property for internal tags defines the tag's data type. For mapped tags, it defines how the manipulated data is to be interpreted by Crimson. The property will be set to a sensible default when the tag is mapped, but can be changed. Note that for most tags, the Treat As property does not have the final say on the actual data type of the tag, as the scaling properties may be used to convert the data further. The following options may be available, depending on the exact data type of the comms data…

TREAT AS	DESCRIPTION
Signed Integer	The data will be treated as a 32-bit signed value, with smaller data values being sign-extended. For example, a 16-bit value of 0x8000 will be converted to 0xFFFF8000.
Unsigned Integer	The data will be treated as a 32-bit signed value, with smaller data items being zero-extended. For example, a 16-bit value of 0x8000 will be converted to 0x00008000. Only available for data items of less than 32 bits in size.
Default Integer	The data will be either zero-extended or sign-extended according to the preference of the communications driver from which the data is obtained. Only available for data items of less than 32 bits in size.
Floating Point	The data will be treated as a 32-bit single-precision floating point value. Only available for data items of exactly 32 bits in size.

- The *Access* property is used for mapped tags to define what kind of communications operations are to be permitted. Internal tags are always set to read and write access, and expression tags are always read-only.

- The *Read Mode* property is used only for array tags. It defines the elements to be read when the array is referenced. The following settings can be used…

READ MODE	DESCRIPTION
Entire Array	All the elements in the array will be read whenever the array is referenced, and access will be blocked until the read operation has completed. This will ensure that data is available, but will produce the slowest performance.
Manual Mode	The array will not be read until a call is made to the `ReadData` function to force a one-time manual update.
On Demand	Array elements will be read as they are referenced. This produces the quickest performance, but stale data may be returned when an element is first accessed.
N Either Side	On Demand operation will be used, but N registers either side of the referenced register will be read as well, thereby making adjacent data available more quickly.

- The *Storage* property is used to indicate whether the tag will be retained through a power-cycle of the target device. This is typically used for internal tags, but mapped write-only tags may also have their values retained.

DATA SCALING

- The *Scaling* property is used for mapped tags to define a final scaling step to be performed on the data. Data may either be scaled to integer or to floating point, irrespective how Crimson is treating the manipulated comms data. For example, an integer value may be scaled to a floating point value, in which case Crimson will consider the tag to be floating point. Likewise, a floating point value might be converted back to an integer, perhaps without even changing its magnitude.

- The *Data From* and *Data To* properties define the domain of the transformation that occurs on read and the range of the transformation that occurs on a write. The values must match the data type specified in Treat As, such that only data that is being treated as floating point can have non-integral values entered in these fields. On read, values beyond these limits are still accepted, and will be scaled to corresponding values beyond the Display limits. In other words, no clipping of the value is performed.

- The *Display From* and *Display To* properties define the range of the transformation that occurs on read and the domain of the transformation that occurs on a write. The values must match the data type specified in Scale To, such that only data that is being scaled to floating point can have non-integral values entered in these fields. On write, values beyond these limits are still accepted, and will be scaled to corresponding values beyond the Data limits. In other words, no clipping of the value is performed.

DATA SIMULATION

- The *Simulate As* property defines the assumed value to be used for the tag when working in the page editor. Entering a sensible value allows a better representation of the page's likely

appearance. This value is also used as the tag's default value by the target device if communication is globally disabled.

DATA ACTIONS

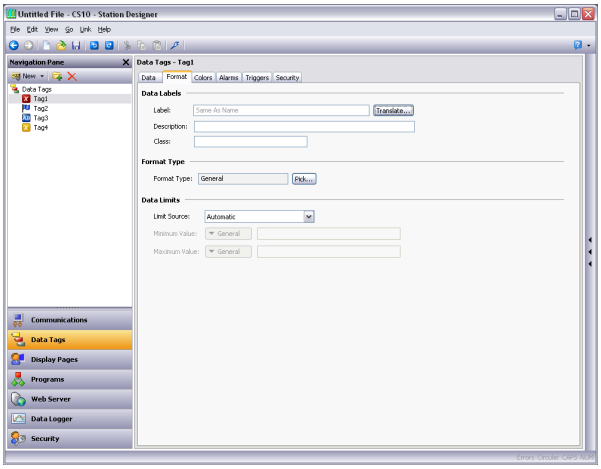
- The *On Write* property defines an action that is to be invoked when the tag is changed. The system variable *Data* will hold the new data value when the write occurs and when the action is executed. The use of On Write properties is covered later in this chapter.

DATA SETPOINT

- The *Use Setpoint* property is used to enable or disable a setpoint for this tag.
- The *SP Value* property defines an expression or another tag that this tag is nominally meant to follow. This setpoint can then be used in alarms or in primitives to implement various functions.

FORMAT PROPERTIES

A numeric tag has the following properties on its Format tab…



DATA LABELS

- The *Label* property was discussed above under Tag Attributes.
- The *Description* property was discussed above under Tag Attributes.
- The *Class* property is reserved for future expansion.

FORMAT TYPE

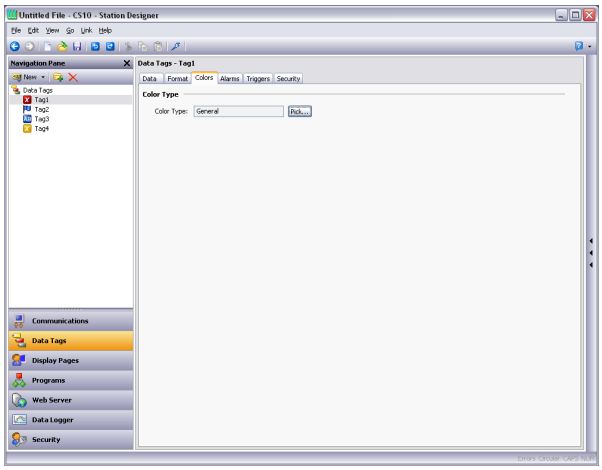
- The *Format Type* property selects the format for this tag. The various types of formats are discussed in detail in a following chapter, as are the other properties that might appear according to the format type that you have selected.

DATA LIMITS

- The *Limit Source* property defines how the tag's data entry limits are defined. The default setting of Automatic results in the Display Range specified on the Data tab being used as a primary source, with the format object being used as a fall-back. If neither source can define a range, the maximum supported range for the tag's data type is used. A setting of From Format can be used to force the format object to be used, while a setting of User Defined can be used to allow manual entry of the limits.
- The *Minimum Value* and *Maximum Value* properties are used to manually define data entry limits when Limit Source is set to User Defined.

COLOR PROPERTIES

A numeric tag has the following properties on its Colors tab…

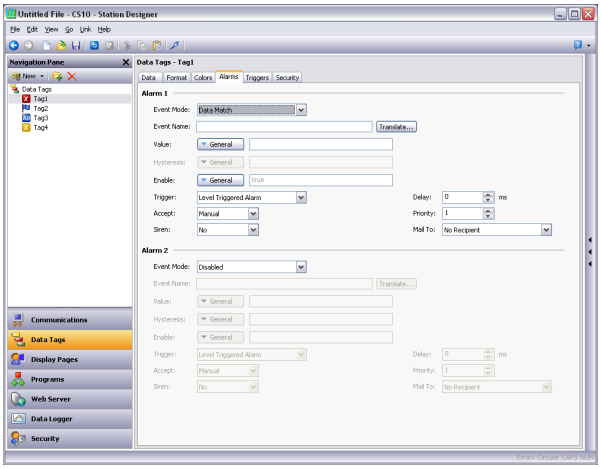


COLOR TYPE

- The *Color Type* property defines the coloring for this tag. The various types of coloring are discussed in detail in a following chapter, as are the other properties that might appear according to the option you have selected.

ALARM PROPERTIES

A numeric tag has the following properties on its Alarms tab…



FOR EACH ALARM

- The *Event Mode* property is used to indicate the logic that will be used to decide whether the alarm should activate. The tables below list the available modes.

MODE	ALARM WILL ACTIVATE WHEN…
Data Match	The value of the tag is equal to the alarm's *Value*.
Data Mismatch	The value of tag is not equal to the alarm's *Value*.
Absolute High	The value of the tag exceeds the alarm's *Value*.
Absolute Low	The value of the tag falls below the alarm's *Value*.
Rise in Value	The value of the tag rises by the alarm's *Value*.
Fall in Value	The value of the tag falls by the alarm's *Value*.
Change in Value	The value of the tag changes by the alarm's *Value*.

The following modes are only available when a setpoint is defined…

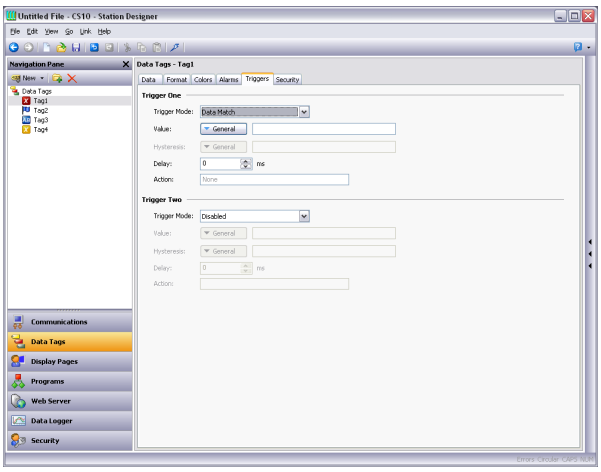
MODE	ALARM WILL ACTIVATE WHEN…
Deviation High	The value of the tag exceeds the tag's *Setpoint* by an amount equal to or greater than the alarm's *Value*.
Deviation Low	The value of the tag falls below the tag's *Setpoint* by an amount equal to or greater than the alarm's *Value*.
Out of Band	The tag moves outside a band, which is equal in width to twice the alarm's *Value* and is centered on the tag's *Setpoint*.
In Band	The tag moves inside a band, which is equal in width to twice the alarm's *Value* and is centered on the tag's *Setpoint*.

- The *Event Name* property defines the name that will be displayed in the alarm viewer or in the event log when referring to this event.

- The *Value* property defines either the absolute value at which the alarm will be activated, the deviation from the setpoint value or the change in value that must occur since the alarm last triggered. The exact interpretation depends on the event mode as described above.
- The *Hysteresis* property is used to prevent an alarm from oscillating between the on and off states when the process is near the alarm condition. For example, for an absolute high alarm, the alarm will become active when the tag exceeds the alarm's value, but will only deactivate when the tag falls below the value by an amount greater than or equal to the alarm's hysteresis. Remember that the property always acts to maintain an alarm once the alarm is activated, and not to modify the point at which the activation occurs.
- The *Enable* property defines an expression that enables or disables the alarm. A non-zero or empty value results in the alarm being enabled, while a zero values results in the alarm being disabled.
- The *Trigger* property is used to indicate whether the alarm should be edge or level triggered. In the former case, the alarm will trigger when the condition specified by the event mode first becomes true. In the latter case, the alarm will remain in the active state while the condition persists. This property can also be used to indicate that this alarm should be used as an event only. In this case, the alarm will be edge triggered, but will not result in an alarm condition. Rather, an event will be logged to internal memory and optionally to CompactFlash.
- The *Delay* property is used to indicate how long the alarm condition must exist before the alarm will become active. In the case of an edge triggered alarm or event, this property also specifies the amount of time for which the alarm condition must no longer exist before subsequent re-activations will result in a further alarm being signaled. As an example, if an alarm is set to activate when a speed switch indicates that a motor is not running even when the motor has been requested to start, this property can be used to provide the motor with time to run-up before the alarm is activated.
- The *Accept* property is used to indicate whether the user will be required to explicitly accept an alarm before it will no longer be displayed. Edge triggered alarms must always be manually accepted.
- The *Priority* property is used to control the order in which alarms are displayed by Crimson's alarm viewer. The lower the numerical value of the priority field, the nearer to the top the alarm will be displayed.
- The *Siren* property is used to indicate whether or not the activation of this alarm should also activate the target device's sounder. While the sounder is active, the panel's display will also flash to better draw attention to the alarm condition.
- The *Mail To* property specifies the email address book entry to which a message should be sent when this alarm is activated. Refer to the Using Services chapter for information on configuring email.
- The *On Accept*, *On Active* and *On Clear* properties are used to specify actions to be executed when the specified change of state occurs. Note all actions will be available, depending on the alarm's trigger mode and accept type.

TRIGGER PROPERTIES

A numeric tag has the following properties on its Trigger tab…



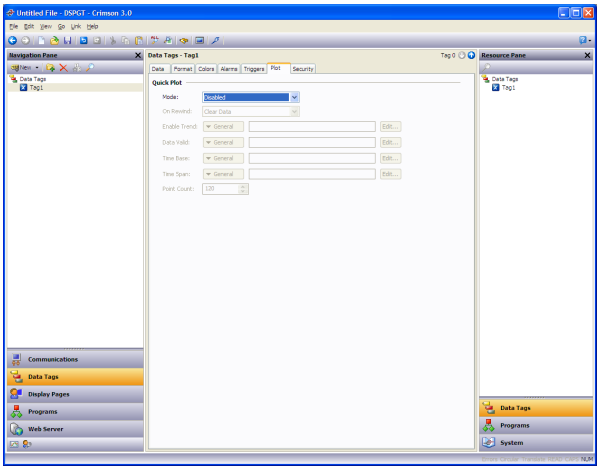
FOR EACH TRIGGER

- The *Trigger Mode* property is as described for the Alarms tab.
- The *Value* and *Hysteresis* properties are as described for the Alarms tab.
- The *Delay* property is as described for the Alarms tab.
- The *Action* property is used to indicate what action should be performed when the trigger is activated. Refer to the Writing Actions chapter for a description of the syntax used to define the various actions that are available.

PLOT PROPERTIES

Quick Plot is a feature added to numeric tags which allows for an easy method of graphically tracking tag values. Once enabled and configured, the tag plot can be added to a display page from the "Core Primitives" category on the Resource Pane. Click and drag the "Quick Plot" primitive onto the desired display page and resize as needed.

The following properties are on the Plot tab…



- The *Mode* property is used to set how data is recorded. *Continuous* mode records in a circular buffer, discarding old values. *Continuous* is the most commonly used mode. *One-shot relative* will start recording when the enable becomes true, and stop when the buffer fills or when enable goes false. The time values used to find the position in the slot will be relative to the time when the plot started. *One-shot absolute* is similar, except that all time values are zero-based.
- *On Rewind* specifies what to do if time goes backwards. This can happen since the time base can be a variable. The options are to either clear the data after the time to which we have stepped back, or shift all the data in the buffer so that the old data is retained but is shifted back in time.
- *Enable* starts and stops the trend.
- *Data Valid* allows gaps to be recorded in the data without stopping the trend and thereby dropping all the data when it restarts.
- By default, *Time Base* is the system time. It is used to define the time base. For specialized applications (e.g. recording ramp-soak performance from an external controller) it can be an external time base.
- *Time Span* is the number of time base ticks to record in the buffer. Note that this is typically larger than the point count and, together with that variable, defines how many ticks each slot will take up.
- *Point Count* is the number of points to store in the buffer. As the quick plot is designed for a basic display of a tag's changes over time, this is typically a smaller value, i.e. smaller than the number of pixels across the display.

SECURITY PROPERTIES

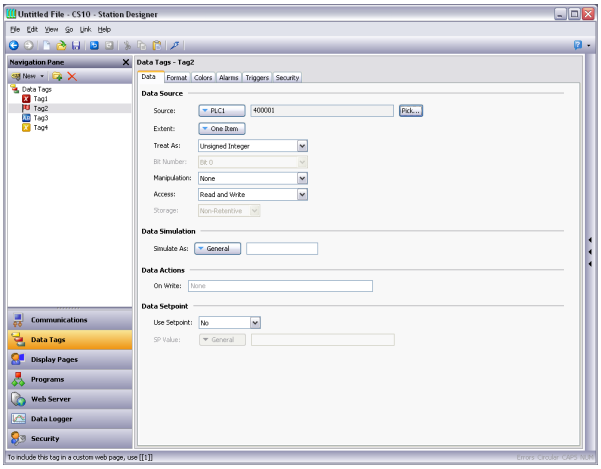
Refer to the Using Security chapter for details of security descriptors.

FLAG TAGS

A flag tag represents one or more on-or-off values and is considered to have an internal data type of integer, no matter what the type of the underlying data. Mapped flag tags allow simple transformations between the raw data and the data that will be used by Crimson.

DATA PROPERTIES

A flag tag has the following properties on its Data tab…



DATA SOURCE

- The *Source* property defines where the tag gets its data. The default setting results in an internal tag, but the drop-down list may be used to select a general expression, another data tag or an item from a remote device.
- The *Extent* property is used to choose between a single element tag or an array. If you select an array, you must enter the required number of elements. Arrays are not permitted for tags whose source is an expression. For mapped tags, the exact number of registers to be read from the remote devices depends upon the type of the registers to which the tag is mapped and the Treat As setting.
- The *Treat As* property is used for mapped tags to define how the on-or-off value is to be derived from the raw comms data and *vice versa*. The following settings may be available, depending on the underlying data type…

TREAT AS	RESULT
Unsigned Integer	The tag will be true if the data is non-zero, or false if it is zero. A true value will be written as an integer value of 1, while a false value will be written as zero. For a mapped array, each array element corresponds to a single comms data element. This setting is available for any comms data of 8 bits or more in size.
Floating Point	The tag will be true if the value is non-zero, or false if it is zero. A true value will be written as a 32-bit floating point value of 1, while a false value will be written as zero. For a mapped array, each array element corresponds to a single comms data element. This setting is available for comms data of exactly 32 bits in size.

TREAT AS	RESULT
Bit Array Little Endian	A single bit is extracted from the data. For a single element, the Bit Number field selects the bit, with the least significant being bit 0. For an array, each element is a single bit, such that the bits are in effect packed within the data items. The first element of the array is the least significant bit, the second is the next-most significant and so on. An 8 element array mapped to a byte data type in a PLC will thus read all 8 bits from a single register.
Bit Array Big Endian	As above, except that the bits are reversed, with a Bit Number field of Bit 0 accessing the most significant bit, and with the first element of an array being sourced from the most significant bit downwards.

- The *Bit Number* property extracts a single bit from multi-bit data items for mapped non-array tags. The property is not used for other configurations.
- The *Manipulation* property defines the transformation that is applied to the tag state after the Treat As logic has been performed when reading data, or before the Treat As logic when writing data. The only option available is to invert the state of the tag. Not a lot else you can do with a single bit value!
- The *Access* property is used for mapped tags to define what kind of communications operations are to be permitted. Internal tags are always set to read and write access, and expression tags are always read-only.
- The *Read Mode* property is used only for array tags. It defines the elements to be read when the array is referenced. The following settings can be used…

READ MODE	DESCRIPTION
Entire Array	All the elements in the array will be read whenever the array is referenced, and access will be blocked until the read operation has completed. This will ensure that data is available, but will produce the slowest performance.
Manual Mode	The array will not be read until a call is made to the `ReadData` function to force a one-time manual update.
On Demand	Array elements will be read as they are referenced. This produces the quickest performance, but stale data may be returned when an element is first accessed.
N Either Side	On Demand operation will be used, but N registers either side of the referenced register will be read as well, thereby making adjacent data available more quickly.

- The *Storage* property is used to indicate whether the tag will be retained through a power-cycle of the target device. This is typically used for internal tags, but mapped write-only tags may also have their values retained.

DATA SIMULATION

- The *Simulate As* property defines the assumed value to be used for the tag when working in the page editor. Entering a sensible value allows a better representation of the page's likely appearance. This value is also used as the tag's default value by the target device if communication is globally disabled.

DATA ACTIONS

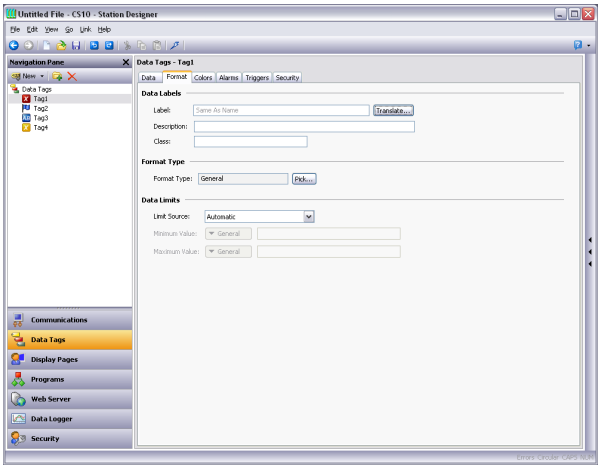
- The *On Write* property defines an action that is to be invoked when the tag is changed. The system variable *Data* will hold the new data value when the write occurs and when the action is executed. The use of On Write properties is covered later in this chapter.

DATA SETPOINT

- The *Use Setpoint* property is used to enable or disable a setpoint for this tag.
- The *SP Value* property defines an expression or another tag that this tag is nominally meant to follow. This setpoint can then be used in alarms or in primitives to implement various functions.

FORMAT PROPERTIES

A flag tag has the following properties on its Format tab…



DATA LABELS

- The *Label* property was discussed above under Tag Attributes.
- The *Description* property was discussed above under Tag Attributes.
- The *Class* property is reserved for future expansion.

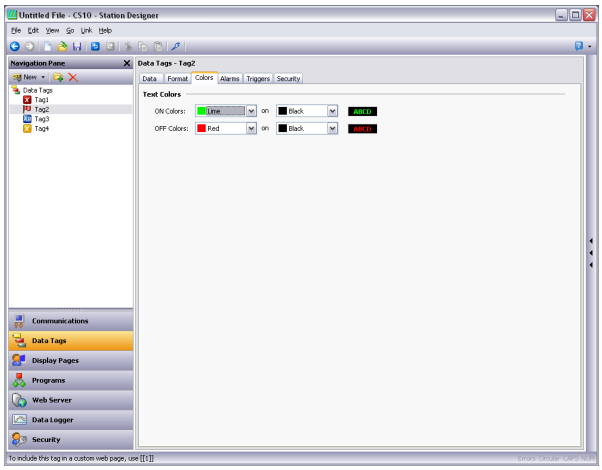
FORMAT TYPE

- The *Format Type* property selects the format for this tag. A Two-State format is used by default, but a Linked format may be substituted instead. The various types of formats are

discussed in detail in a following chapter, as are the other properties that might appear according to the format type that you have selected.

COLOR PROPERTIES

A numeric tag has the following properties on its Colors tab…

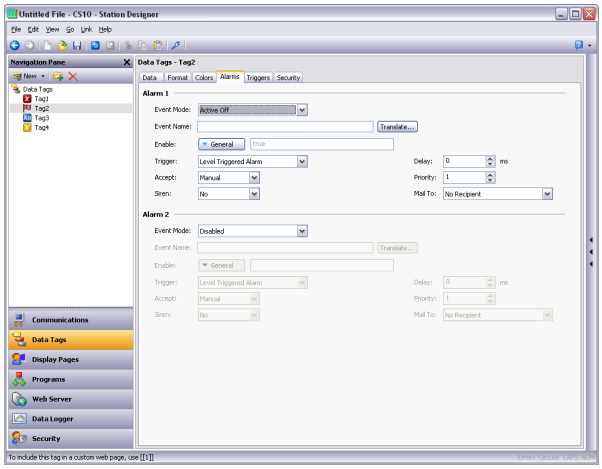


COLOR TYPE

- The *Color Type* property defines the coloring for this tag. A Two-State coloring is selected by default, but a General, Linked or Fixed coloring may be substituted. The various colorings are discussed in detail in a later chapter, as are the other properties that might appear according to the option you have selected.

ALARM PROPERTIES

A flag tag has the following properties on its Alarms tab…



FOR EACH ALARM

- The *Event Mode* property is used to indicate the logic that will be used to decide whether the alarm should activate. The tables below list the available modes.

MODE	ALARM WILL ACTIVATE WHEN...
Active On	The tag is true.
Active Off	The tag is false.
Change of State	The tag changed.

The following modes are only available when a setpoint is defined…

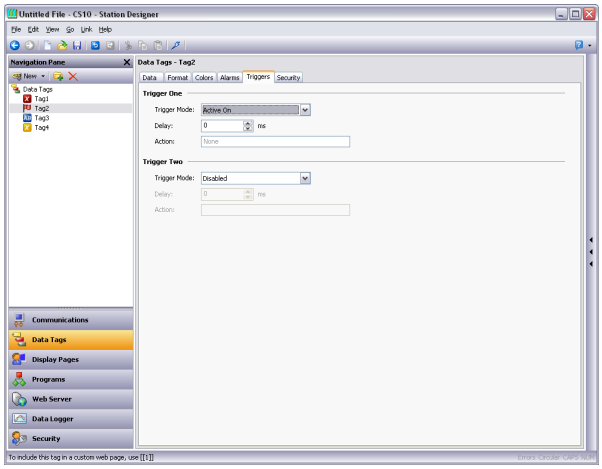
MODE	ALARM WILL ACTIVATE WHEN...
Not Equal to SP	The tag does not equal its setpoint.
Off When SP On	The tag does not respond to an ON setpoint.
On When SP Off	The tag does not respond to an OFF setpoint.
Equal to SP	The tag equals its setpoint.

- The *Event Name* property defines the name that will be displayed in the alarm viewer or in the event log as appropriate. Crimson will suggest a default name based upon the tag's label, and the event mode that has been selected.
- The *Enable* property defines an expression that enables or disables the alarm. A non-zero or empty value results in the alarm being enabled, while a zero values results in the alarm being disabled.
- The *Trigger* property is used to indicate whether the alarm should be edge or level triggered. In the former case, the alarm will trigger when the condition specified by the event mode first becomes true. In the latter case, the alarm will remain in the active state while the condition persists. This property can also be used to indicate that this alarm should be used as an event only. In this case, the alarm will be edge triggered, but will not result in an alarm condition. Rather, an event will be logged to internal memory and optionally to CompactFlash.
- The *Delay* property is used to indicate how long the alarm condition must exist before the alarm will become active. In the case of an edge triggered alarm or event, this property also specifies the amount of time for which the alarm condition must no longer exist before subsequent re-activations will result in a further alarm being signaled. As an example, if an alarm is set to activate when a speed switch indicates that a motor is not running even when the motor has been requested to start, this property can be used to provide the motor with time to run-up before the alarm is activated.
- The *Accept* property is used to indicate whether the user will be required to explicitly accept an alarm before it will no longer be displayed. Edge triggered alarms must always be manually accepted.
- The *Priority* property is used to control the order in which alarms are displayed by Crimson's alarm viewer. The lower the numerical value of the priority field, the nearer to the top the alarm will be displayed.
- The *Siren* property is used to indicate whether or not the activation of this alarm should also activate the target device's sounder. While the sounder is active, the panel's display will also flash to better draw attention to the alarm condition.

- The *Mail To* property specifies the email address book entry to which a message should be sent when this alarm is activated. Refer to the Using Services chapter for information on configuring email.
- The *On Accept*, *On Active* and *On Clear* properties are used to specify actions to be executed when the specified change of state occurs. Note all actions will be available, depending on the alarm's trigger mode and accept type.

TRIGGER PROPERTIES

A flag tag has the following properties on its Trigger tab…



FOR EACH TRIGGER

- The *Trigger Mode* property is as described for the Alarms tab.
- The *Delay* property is as described for the Alarms tab.
- The *Action* property is used to indicate what action should be performed when the trigger is activated. Refer to the Writing Actions chapter for a description of the syntax used to define the various actions that are available.

SECURITY PROPERTIES

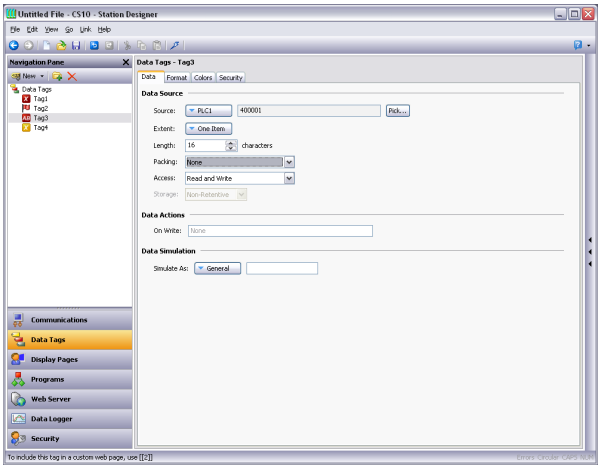
Refer to the Using Security chapter for details on security descriptors.

STRING TAGS

A string tag represents one or more strings of Unicode characters. While Crimson 3 works entirely in Unicode, it can read and write strings from 8-bit sources, too. Mapped string tags support various encodings, allowing one or more characters to be extracted from one register.

DATA PROPERTIES

A string tag has the following properties on its Data tab…



DATA SOURCE

- The *Source* property defines where the tag gets its data. The default setting results in an internal tag, but the drop-down list may be used to select a general expression, another data tag or an item from a remote device.
- The *Extent* property is used choose between a single element tag or an array. If you select an array, you must enter the required number of elements. Arrays are not permitted for tags whose source is an expression. For mapped tags, the exact number of registers to be read from the remote devices depends upon the type of the registers to which the tag is mapped, the length and the Packing setting.
- The *Length* property defines the length of the string. Non-retentive internal strings do not have to have a length defined, as they can store a string of any reasonable length.
- The *Packing* property is used for mapped tags to define how the Unicode string value is to be derived from the raw comms data and *vice versa*. The following settings may be available, depending on the underlying data type…

PACKING	RESULT
None	Each comms data item is used to source a single character for the string. 8-bit values will be treated as ASCII, while 16-bit and larger values will be treated as Unicode.
ASCII Big Endian	Each 8-bit unit in the data item is used to source a single ASCII character, with the most significant 8 bits being used for the first character. Only available for data items of 16 bits or greater in size.
ASCII Little Endian	Each 8-bit unit in the data item is used to source a single ASCII character, with the least significant 8 bits being used for the first character. Only available for data items of 16 bits or greater in size.

PACKING	RESULT
Unicode Big Endian	Each 16-bit unit in the data item is used to source a single Unicode character, with the most significant 16 bits being used for the first character. Only available for data items of 32 bits in size.
Unicode Little Endian	Each 16-bit unit in the data item is used to source a single Unicode character, with the least significant 16 bits being used for the first character. Only available for data items of 32 bits in size.
Hex String Little Endian	Each 4-bit unit in the data item is used to source a single hex character in the range '0'-'9' and 'A'-'F', with the least significant 4 bits being used first. Writes to strings with this packing method are not supported.
Hex String Big Endian	Each 4-bit unit in the data item is used to source a single hex character in the range '0'-'9' and 'A'-'F', with the most significant 4 bits being used first. Writes to strings with this packing method are not supported.

- The *Access* property is used for mapped tags to define what kind of communications operations are to be permitted. Internal tags are always set to read and write access, and expression tags are always read-only.
- The *Read Mode* property is used only for array tags. It defines the elements to be read when the array is referenced. The following settings can be used…

READ MODE	DESCRIPTION
Entire Array	All the elements in the array will be read whenever the array is referenced, and access will be blocked until the read operation has completed. This will ensure that data is available, but will produce the slowest performance.
Manual Mode	The array will not be read until a call is made to the `ReadData` function to force a one-time manual update.
On Demand	Array elements will be read as they are referenced. This produces the quickest performance, but stale data may be returned when an element is first accessed.
N Either Side	On Demand operation will be used, but N registers either side of the referenced register will be read as well, thereby making adjacent data available more quickly.

- The *Storage* property is used to indicate whether the tag will be retained through a power-cycle of the target device. This is typically used for internal tags, but mapped write-only tags may also have their values retained.

DATA SIMULATION

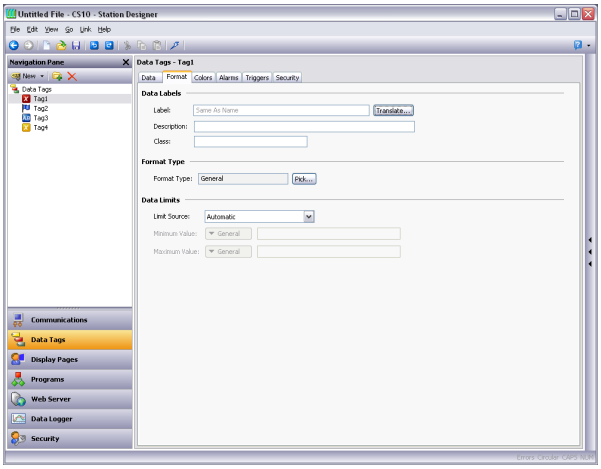
- The *Simulate As* property defines the assumed value to be used for the tag when working in the page editor. Entering a sensible value allows a better representation of the page's likely appearance. This value is also used as the tag's default value by the target device if communication is globally disabled.

DATA ACTIONS

- The *On Write* property defines an action that is to be invoked when the tag is changed. The system variable *Data* will hold the new data value when the write occurs and when the action is executed. The use of On Write properties is covered later in this chapter.

FORMAT PROPERTIES

A string tag has the following properties on its Format tab…



DATA LABELS

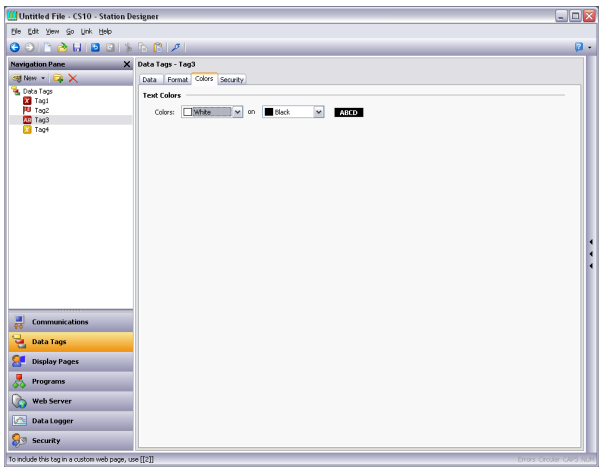
- The *Label* property was discussed above under Tag Attributes.
- The *Description* property was discussed above under Tag Attributes.
- The *Class* property is reserved for future expansion.

FORMAT TYPE

- The *Format Type* property selects the format for this tag. A String format is used by default, but a General or Linked format may be substituted. The various types of formats are discussed in detail in a following chapter, as are the other properties that might appear according to the format type that you have selected.

COLOR PROPERTIES

A string tag has the following properties on its Colors tab…



COLOR TYPE

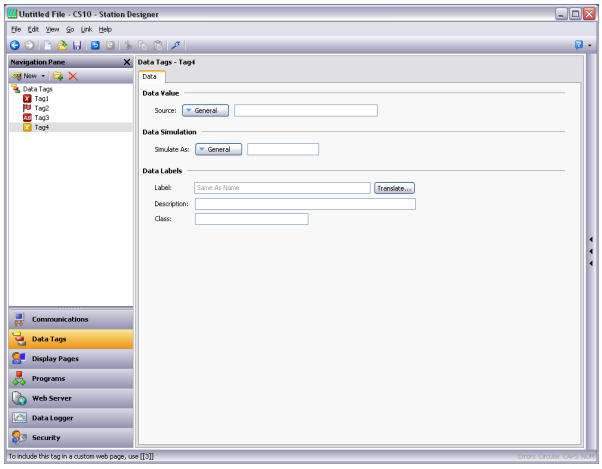
- The *Color Type* property defines the coloring for this tag. A Fixed coloring is selected by default, but a General or Linked coloring may be substituted. The various colorings are discussed in detail in a later chapter, as are the other properties that might appear according to the option you have selected.

SECURITY PROPERTIES

Refer to the Using Security chapter for details on security descriptors.

BASIC TAGS

Basic tags are used to represent constants or expressions…



DATA VALUE

- The *Data Value* property defines the value of the tag. It must be an expression. The tag itself will adopt the data type of the expression that is used.

DATA SIMULATION

- The *Simulate As* property defines a value to be used as the default for the tag when editing display pages. Entering a sensible value allows a better representation of the page's likely appearance. This value is also used as the tag's default value by the target device if communication is globally disabled.

DATA LABELS

- The *Label* property was discussed above under Tag Attributes.
- The *Description* property was discussed above under Tag Attributes.
- The *Class* property is reserved for future expansion.

ADVANCED TOPICS

ARRAY PROPERTIES

Many of the properties of array tags can be made variable based upon the exact element of the array being referenced. To achieve this, a system property called `i` is set to the element index during the evaluation of those properties. For example, the Label property could be set to `="Element " + AsText(i+1)` to label the array elements Element 1, Element 2 etc.

This feature can be used with the following properties…

- The tag's label.
- The tag's scaling values.
- The tag's setpoint.
- The tag's limits.
- The tag's On Write property.
- The tag's event labels.
- The tag's event and trigger values and hysteresis settings.
- The tag's trigger actions.

Note that triggers and events are evaluated separately for each element of the array for which they are configured, allowing several events or triggers to be created at once. The only limitation to this feature is that alarms and events only operate for the first 256 elements of the array. Triggers operate for all elements, regardless of the size of the array.

TAG DATA FLOW

As you will have noticed, numeric tags in particular have a number of data transformations that occur between the comms data and the value actually used by Crimson. These can be configured to handle just about any sort of data in any way you like, but the exact way in which they operate for numeric tags deserves further attention.

NUMERIC TAG READ PROCESS

When data is read from a device, the following steps occur…

- The comms driver is asked to read a value based on the address setting that has been defined for the source of the tag. Based on the type of the address, the driver may combine more than one register to create the data value. For example, reading a single Word as Long value will result in two registers being read and combined by the driver using its knowledge of the device's word ordering.
- The comms data is then modified according to the Manipulation property for the tag in question. These processes perform bit or byte level changes to the data, typically to account for driver incompatibilities or other situations where the data is not in the form that the comms driver normally expects to encounter.
- The manipulated data is then interpreted in conjunction with the tag's Treat As property, being viewed as a 32-bit integer or a 32-bit single-precision floating-point value as appropriate. Data items smaller than 32 bits will be either zero- or sign-extended based per the configuration. If no scaling is defined, the result of this step defines the final value and data type of the tag.
- If scaling is defined, the interpreted data is then scaled according to the domain and range defined for the tag. The result of the scaling may be of a different type from the interpreted data, such that a floating point value may be scaled to an integer or *vice versa*. Assuming scaling is defined, the result of this step then defines the final value and data type of the tag.

NUMERIC TAG WRITE PROCESS

When data is written to a device, the following steps occur…

- If scaling is defined, the domain and range are reversed, converting the data back to an unscaled value whose data type is defined by the Treat As property.
- If the unscaled data is larger than the comms data, the high-order bits are removed, producing a stripped version of the data suitable for the next step.
- The stripped data is then modified according to the Manipulation property, reversing the transformation applied above, producing comms data.
- The comms driver than takes the comms data, and writes it to one or more registers in the target device according to the type of the address.

USING ON WRITE

A tag's On Write property contains an action which is executed when a change is made to the tag. While the action is being executed, a system property called `Data` is set to the new value, allowing the new data to be examined. There are three typical uses for this feature…

- Regular read-and-write tags can have an On Write property defined to allow some action to be taken on demand. For example, a database may need to store the value of a tag in two formats, one being the original tag format and the other being a transformed version. While

there are other ways of doing this, one method is to use the On Write property to catch the write, and then run a program to calculate and store the transformed version.

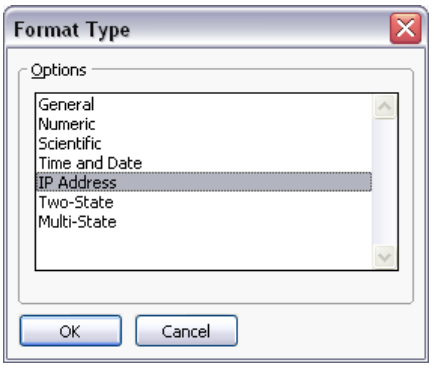
- Read-only tags can be made writable by defining an On Write property. While this seems odd, imagine, for example, that a PID loop has a read-only property to indicate its current output power, and a read-write property to define the manual output power. You could define display fields to allow data entry to the output power when in manual mode, and catch them using the On Write property, thereby writing the values to the manual output power.
- Complex transformations can be implemented by defining an expression tag to perform the forward transformation and an On Write action to perform the inverse. For example, a tag could be set to `Sqrt([40001])` to take the square root of a value in a Modbus PLC. Since this is an expression tag, it is by definition read-only, but writes can be allowed by defining an On Write equal to `[40001] = Data*Data`, thereby reversing the square root calculation.

USING FORMATS

Numeric tags can have one of various data formats selected, while flag and string tags have their formats fixed to Two-State and General, respectively. Each format type will take a data value and convert it to or from a text string.

FORMAT TYPES

The following formats are supported…



- *General* format provides simple formatting of values, converting numeric values to signed decimal values, and passing on strings without further processing. The general format has no configuration properties, and is the default format for string tags. It is also implicitly used by basic tags.
- *Linked* format uses the data format of another tag to format the tag that you are configuring. It is useful for creating format templates and then applying them to many tags in the same database. This can avoid repetition, and make it easier to adjust settings such as units or decimal point counts.
- *Numeric* format takes a floating point or integer value and converts it to a string, using a specific number base and selecting the required number of digits before and after the decimal point. It can also add a prefix string and a units string to the value, and handle signed or unsigned values.
- *Scientific* format takes a floating point or integer value and converts it to exponential format, selecting the required number of digits after the decimal point. It can also add a prefix string and a units string to the value.
- *Time and Date* format takes an integer value and treats it as a number of seconds elapsed since 1st January 1997. It can display the result as a date value, a time value or both, or treat the value as elapsed time that can contain more than 24 in its hours value. Date formatting and time formatting options are supported to allow for various international standards.
- *IP Address* format takes an integer value and displays it as four decimal bytes separated by periods. This allows a 32-bit number to be displayed as an IP address without further configuration.

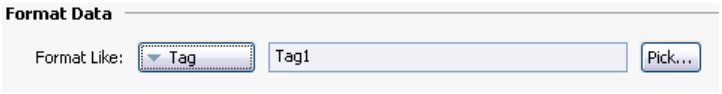
- *Two-State* format takes a numeric value and displays one of two strings based on whether the value is zero or non-zero. This is the permanently defined format type for flag tags.
- *Multi-State* format takes a numeric value and compares it against a table containing values and strings. Either the string associated with a matching data value is displayed, or the format can be configured to display the last string with a value not higher than that string's associated data.
- *String* format takes a string value and either restricts it length during input, or applies a template that indicates what type of character can be entered at which point in the string. This allows, for example, a string to be formatted as United State telephone number, with the parentheses and dashed being inserted upon display without the need to store them in the string data.

GENERAL FORMAT

General format has no properties.

LINKED FORMAT

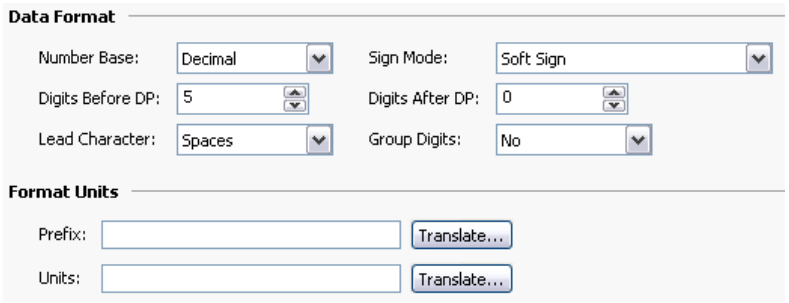
Linked format has the following properties…



- The *Format Like* property is used to select a tag from which the formatting information for this tag is to be obtained. For correct operation, a tag of the correct data type should be used, such that, for example, a string tag's format should be based upon another string tag. Failure to observe this requirement will result in a fall-back to the default formatting rules.

NUMERIC FORMAT

Numeric format has the following properties…



DATA FORMAT

- The *Number Base* property defines the radix of the displayed value. The Passcode setting works in decimal but masks the digits using an asterisk. Many of the other options will be disabled when a non-decimal mode is used.

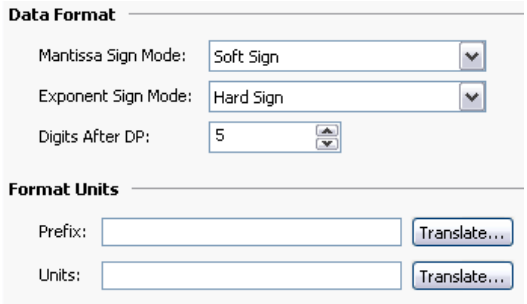
- The *Sign Mode* property defines how the data is treated, and how the sign is displayed. A value of Unsigned will display the value as a 32-bit unsigned number, thereby allowing such values to be displayed and entered, even though Crimson cannot perform any math on values that will not fit in a 32-bit signed representation. A value of Soft Sign will display a leading minus sign for negative numbers and a space for positive numbers, while a value of Hard Sign will display a leading plus sign rather than the space.
- The *Digits Before DP* property defines the number of digits to be shown before the decimal point. For values without decimals, this is the total number of digits to be shown and therefore controls the size of the data field.
- The *Digits After DP* property defines the number of digits to be shown after the decimal point. For integer values, the decimal point is inserted into the integer representation, such that 1234 would be displayed and entered as 12.34 if this property were set to two. A value of zero suppresses the decimal point.
- The *Lead Character* property defines how values with leading zeroes are formatted. Leading zeroes may either be retained, replaced with spaces or removed completely. Removing them can sometimes cause values on a display to show unpleasant jitter as they change their number of digits, particularly if the value is centered within a field.
- The *Group Digits* property enables the insertion of comma separators every three digits for decimal numbers, with analogous behavior for other number bases.

FORMAT UNITS

- The *Prefix* property defines a string to be displayed before the numeric value.
- The *Units* property defines a string to be displayed after the numeric value.

SCIENTIFIC FORMAT

Scientific format has the following properties…



DATA FORMAT

- The *Mantissa Sign Mode* property defines how the sign is displayed on the mantissa. A value of Soft Sign will display a leading minus sign for negative numbers and a space for positive numbers, while a value of Hard Sign will display a leading plus sign rather than the space.

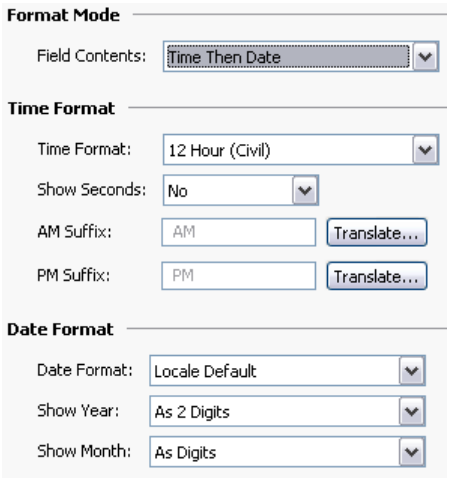
- The *Exponent Sign Mode* property defines how the sign is displayed on the exponent. A value of Soft Sign will display a leading minus sign for negative values and nothing for positive values, while a value of Hard Sign will display a leading plus sign for positive values instead.
- The *Digits After DP* property defines the number of digits to be shown after the decimal point. By definition, there is always one digit before the decimal in scientific format. A value of zero suppresses the decimal point.

FORMAT UNITS

- The *Prefix* property defines a string to be displayed before the numeric value.
- The *Units* property defines a string to be displayed after the numeric value.

TIME AND DATE FORMAT

Time and Date format has the following properties…



FORMAT MODE

- The *Format Mode* property is used to indicate whether the field should display the time, the date or both. In the last case, this property also indicates in which order the two elements should be shown. Options are also provided to allow a time value to be treated as an elapsed period of time, rather than a time that is paired with a date. For example, a value of 25.5 hours will display as 25:30 in an elapsed mode. In a conventional time mode, it will display 00:30, as the system will assume a time early in the morning on 2nd January 1997.

TIME FORMAT

- The *Time Format* property is used to indicate whether 12-hour (civil) or 24-hour (military) time format should be used. As with other properties, leaving this set to Locale Default will allow Crimson to pick a suitable format according to the language selected within the operator panel.

- The *Time Separator* property is used to select the character that will be placed between the elements of the time display. The default value will be based upon the current language selection, but can be overridden as required.
- The *AM Suffix* and *PM Suffix* properties are used with 12-hour mode to indicate the text to be appended to the time field in the morning and afternoon as appropriate. If you leave the property undefined, Crimson will use a default.
- The *Show Seconds* property is used to indicate whether the time field should include the seconds, or whether it should just comprise hours and minutes.

DATE FORMAT

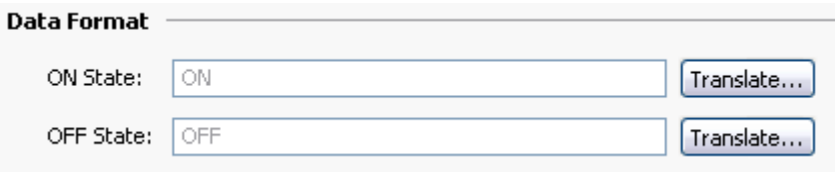
- The *Date Format* property is used to indicate the order in which the various date elements (i.e. date, month and year) should be displayed.
- The *Date Separator* property is used to select the character that will be placed between the elements of the date display. The default value will be based upon the current language selection, but can be overridden as required.
- The *Show Year* property is used to indicate whether the date field should include the year, and if so, how many digits should be shown for that element.
- The *Show Month* property is used to indicate whether the month should be displayed as digits (i.e. 01 through 12) or as its short name (i.e. Jan though Dec).

IP ADDRESS FORMAT

IP Address format has no properties.

TWO-STATE FORMAT

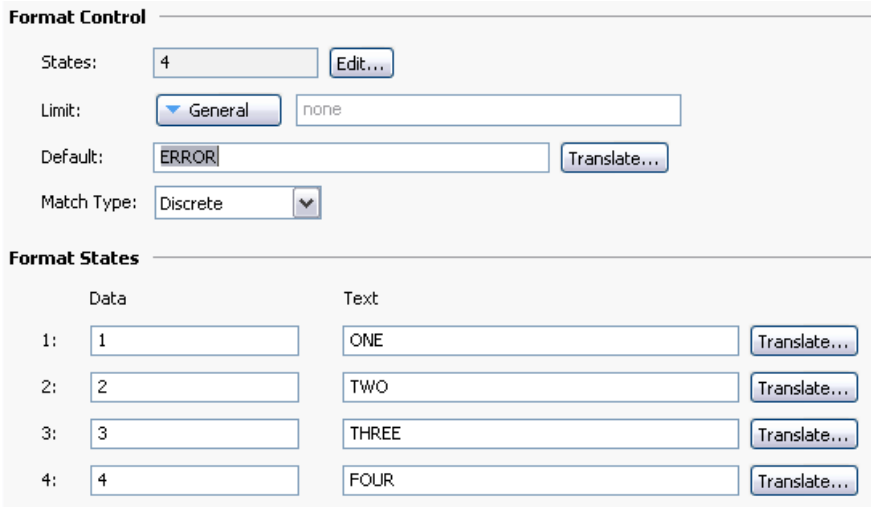
Two-State format has the following properties…



- The *ON State* property defines the text to be shown if the value is non-zero.
- The *OFF State* property defines the text to be shown if the value is zero.

THE MULTI-STATE FORMAT

The Multi-State format has the following properties…



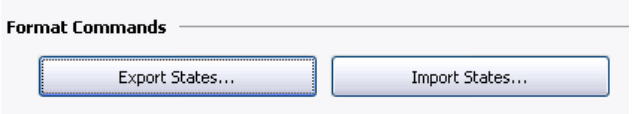
FORMAT CONTROL

- The *States* property defines how many states the multi-state format will contain, up to a maximum of 500 entries. The window displaying the format will update to show the required number of Data and Text properties.
- The *Limit* property defines how many states will be used when matching data against this format. It can be dynamically adjusted, while the absolute number of states is statically defined. This property is useful when the state fields are populated at runtime, as it allows unused fields to be skipped during the data entry process.
- The *Default* property defines a string to be displayed if the data cannot be matched against the defined states. If no value is provided, the numeric representation of the unmatched state will be displayed in parentheses.
- The *Match Type* property defines how the data is compared against the various states. If Discrete is selected, the tag data must match a given state's data value in order for that state to be used. If Ranged is selected, Crimson assumes that the state data values are in increasing numerical order, and will use a state value if the tag data is less than or equal to that state's data value but greater than the prior state's data value. During data entry, ranged format objects assign values equal to the individual states' actual data values.

FORMAT STATES

- The *Data* and *Text* properties define the data value and display text for each state in this format. States with empty text fields are disabled and are ignored.

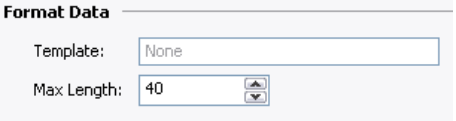
FORMAT COMMANDS



Multi-state format objects also provide buttons to allow their various states and the associated properties to be exported to or imported from Unicode text files. These files can then be edited by an application such as Microsoft Excel.

THE STRING FORMAT

The String Format has the following properties…



- The *Template* property is used to enter an optional "picture" of what the string should look like. A template comprises a number of special formatting characters that indicate what type of character is acceptable at that position. The following formatting characters are supported…

Character In Template	Permitted Characters A-Z	a-z	0-9	Space	Misc
A	Yes	-	-	-	-
a	Yes	Yes	-	-	-
S	Yes	-	-	Yes	-
s	Yes	Yes	-	Yes	-
N	Yes	-	Yes	-	-
n	Yes	Yes	Yes	-	-
M	Yes	-	Yes	Yes	-
m	Yes	Yes	Yes	Yes	-
0	-	-	Yes	-	-
X	Yes	Yes	Yes	Yes	Yes

The additional characters referred to by the "Misc" column are…

```
,.:;+-=!?%/$
```

Any characters that are not formatting characters are interpreted as literals and displayed without their having to be present in the underlying data. For example, a template of "(000) 000-0000" will allow entry of US standard telephone numbers without the user having to enter the punctuation, and without those characters having to be stored for each string.

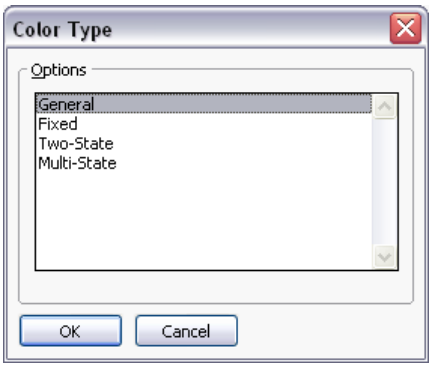
- The *Max Length* property may be used as an alternative to the Template property to allow free-form entry to a maximum number of characters. Note that the format length and the underlying data length are independent values, but that the former should not typically be larger than the latter.

USING COLORINGS

Numeric tags can have one of various so-called colorings selected, while flag and string tags have their colors fixed to Two-State and General, respectively. Each coloring will take a data value and convert to a foreground and background color pair.

TYPES OF COLORING

The following colorings are supported…



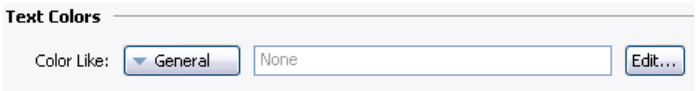
- The *General* coloring always returns white on black.
- The *Linked* format uses the coloring of another tag to format the tag that you are configuring. It is useful for creating templates and then applying them to many tags in the same database. This can avoid repetition, and make it easier to adjust your color settings from a single location.
- The *Fixed* coloring always returns a fixed pair of colors.
- The *Two-State* coloring takes a numeric value and picks one of two color pairs based on whether the value is zero or non-zero. This is the permanently defined coloring for flag tags.
- The *Multi-State* coloring takes a numeric value and compares it against a table containing data values and color pairs. Either a color pair associated with a matching data value is selected, or the selector can be configured to use the last color pair with an associated data value not higher than the data.

GENERAL COLORING

The General coloring has no properties.

LINKED COLORING

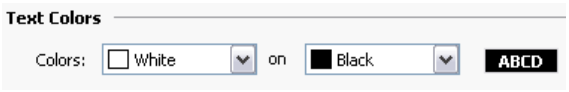
Linked coloring has the following properties…



- The *Color Like* property is used to select a tag from which the coloring information for this tag is to be obtained. For correct operation, a tag of the correct data type should be used, such that, for example, a numeric tag's coloring should be based upon another numeric tag. Failure to observe this requirement will result in a fall-back to the default formatting rules.

FIXED COLORING

The Fixed coloring has the following properties…



- The *Colors* property defines the colors to be used all the time.

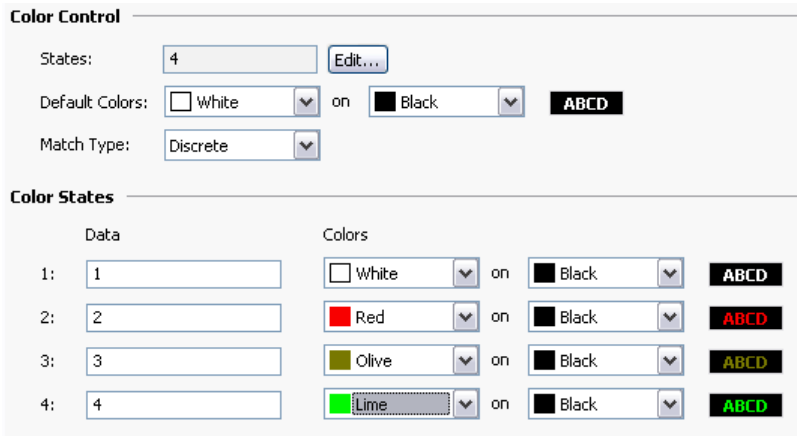
TWO-STATE COLORING

The Two-State coloring has the following properties…

- The *ON Colors* property defines the colors to be used when the tag is non-zero.
- The *OFF Colors* property defines the colors to be used when the tag is zero.

MULTI-STATE COLORING

The Multi-State coloring has the following properties…



FORMAT CONTROL

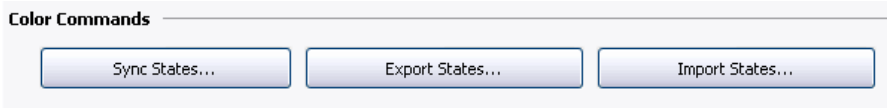
- The *States* property defines how many states the multi-state selector will contain, up to a maximum of 500 entries. The window displaying the selector will update to show the required number of Data and Text properties.
- The *Default Colors* property defines the colors to be used if the data cannot be matched against the defined states.

- The *Match Type* property defines how the data is compared against the various states. If Discrete is selected, the tag data must match a given state's data value in order for that state to be used. If Ranged is selected, Crimson assumes that the state data values are in increasing numerical order and will use a state value if the tag data is less than or equal to that state's data value but greater than the prior state's data value.

FORMAT STATES

- The *Data* and *Colors* properties define the data and color values for each state.

COLOR COMMANDS



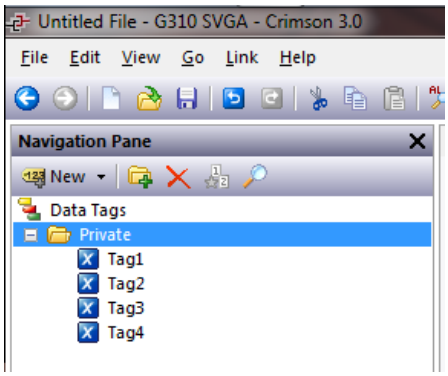
Multi-state coloring objects also provide buttons to allow their states and the associated properties to be exported to or imported from Unicode text files. These files can then be edited by an application such as Microsoft Excel. An additional button allows the Data fields of the coloring to be synchronized with the Data fields of a Multi-State format object configured for the same tag, avoiding your having to enter the same values twice.

PASSWORD PROTECTING TAGS

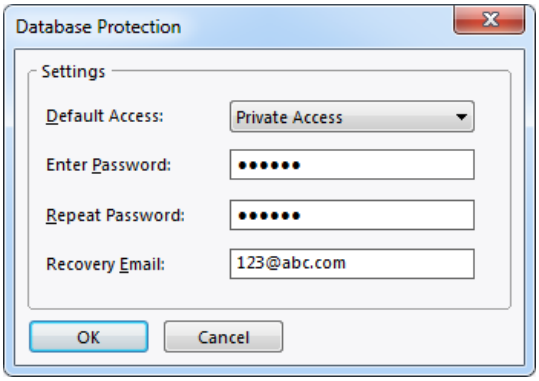
Tags can be hidden from view and password protected by placing them in a folder named "Private."

PROTECTING DATA TAGS

Create a single folder named "Private" and place all data tags to be protected into this folder. Subfolders under the "Private" folder can be created and named as desired and they will be protected as well.



On the menu bar, select "File" then "Protection. In the dialog box, select "Private Access" and then enter the required information. After protection is complete, the "Private" folder can be seen, but data tags will be hidden. To unprotect data tags, select "File" then "Protection." Enter the password and then select "Full Access" in the dropdown menu.

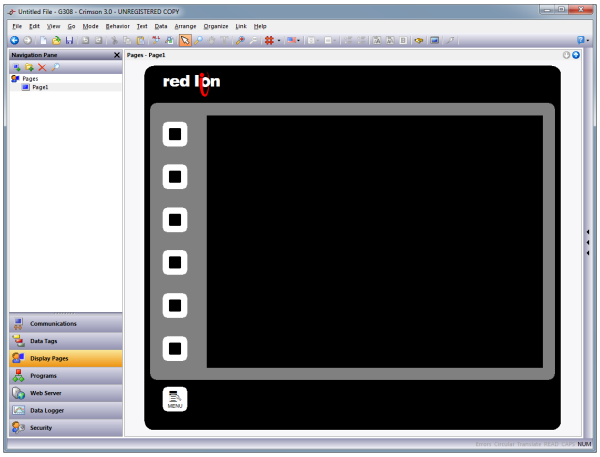


CREATING DISPLAY PAGES

Selecting the Display Pages category in the Navigation Pane gives access to the new Crimson graphics editor. This editor is designed to allow the quick and efficient creation of attractive displays, while also providing the maximum flexibility.

EDITOR BASICS

The graphics editor is shown below in its initial state…



The Editing Pane shows a representing of the target device, including both the keys and the display area itself. At the lowest zoom level, the entire panel will be shown, even if this means allocating less than one pixel on your PC's display for each pixel on the display of the target device. In this situation, pages can still be viewed and most editing can be performed, but accuracy will be somewhat reduced. A warning message to that effect is thus displayed.

WORKING WITH PAGES

Manipulation of display pages via the Navigation List is intuitive, and operates as for any other item in a Crimson database. That said, it is worth reiterating the fact that pages can be copied between databases by simply selecting them in one database's Navigation Pane and dragging them to the corresponding category in the target database. This makes it very easy to build new databases by combining previously used page designs.

CHANGING THE ZOOM LEVEL

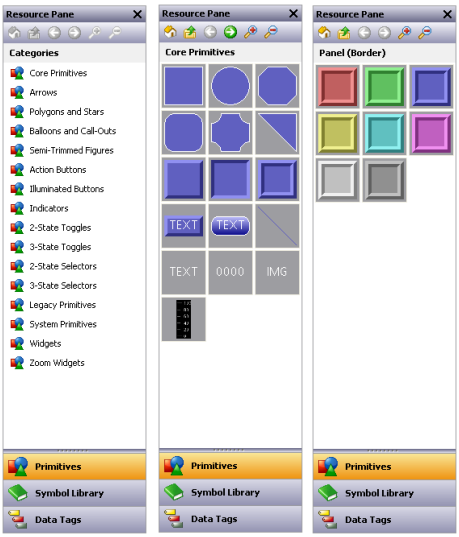
Zooming in and out is most easily performed using the mouse wheel. If you do not have such a mouse, you can use the editor's zoom mode by selecting the magnifying glass from the toolbar. In this mode, left-clicks will zoom in, and either right-clicks or left-clicks with **CTRL** held down will zoom out. You may also use the zoom commands on the View menu.

The first zoom step will take you from the full-panel view to a 1:1 display, centering the target device's display in your editing window. Thereafter, zooming is performed so as to keep the data under your mouse pointer in view, thereby making it easier to choose which area of the display you wish to examine in more detail.

THE RESOURCE PANE

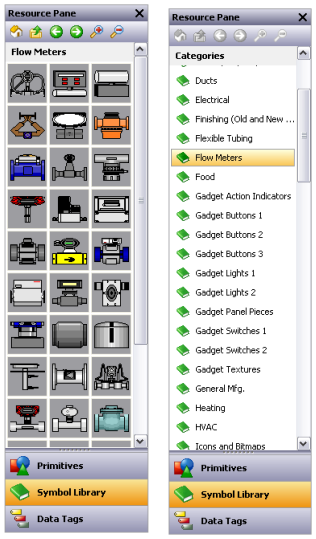
Display pages are typically built from items dragged from the Resource Pane. You can either slide out the Resource Pane by clicking the arrowed bar to the right-hand side of the window, or you can choose to lock the Resource Pane in place, perhaps maximizing your window to increase your available workspace. The Resource Pane has three categories…

PRIMITIVES



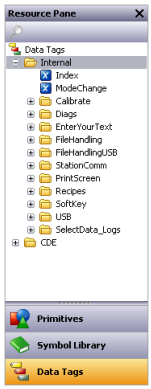
The Primitives category is used to access the key building blocks used to assemble display pages, and is shown to the left in its various states. You will notice that the top-level contains a number of sub-categories, each of which provides access to a number of primitives. Clicking on an icon displays a sub-category and its primitives. Clicking on a given primitive displays versions of that primitive in predefined colors. The icons in the toolbar can be used to move between sub-categories, to move up to a higher level or to change the number of primitives displayed per row. The primitives are described in the next chapter.

SYMBOL LIBRARY



The Symbol Library category operates in a manner that is very similar to the Primitives category, providing access to a number of sub-categories, each of which contains a number of predefined symbols. Clicking on a given symbol provides a number of pre-colored versions of that symbol, although this facility is used less often than it is with primitives. Take some time to explore the Symbol Library—it contains many thousands of possible images, and its correct use can produce more attractive and easy-to-use databases.

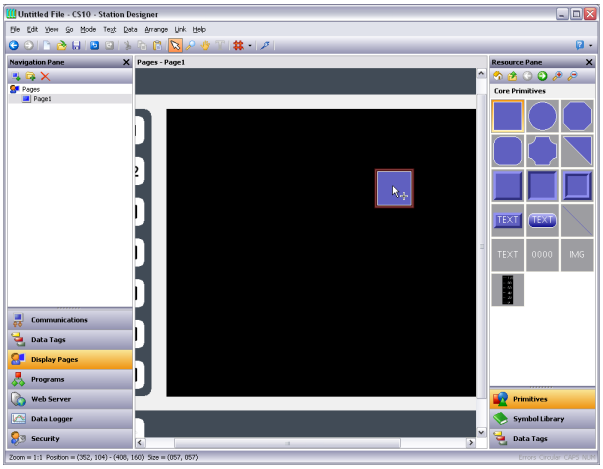
DATA TAGS



The Data Tags category contains a tree view of all the data tags in the current database. It is used both to drag tags directly on to a display page, and to provide access to tags while configuring the primitive properties. Dragging a tag onto a page will create a data box that is bound to that tag, with all the formatting properties based on the properties defined by the tag itself. You may also select and drag multiple tags by using the **SHIFT** and **CTRL** keys in the usual way. These facilities make it very quick and easy to add data to a page.

ADDING ITEMS TO A PAGE

As mentioned above, the various items in the Resource Pane can be dragged onto the editor, thereby adding them to a display page. Suitable primitives will be created for tags and images. The example below shows how, after clicking on the Core Primitives selection in the Primitives category, a rectangle primitive can be dragged onto the page…



WORKING WITH PRIMITIVES

The following sections describe how to perform common operations on primitives.

SELECTING PRIMITIVES

To select a display primitive, simply move your mouse pointer over the primitive in question, and perform a left-click. You will notice that while your pointer is hovering over a primitive, a bounding rectangle is drawn in blue to help show what will be selected. When the actual selection is performed, the rectangle will change to red, and handles will appear, so as to allow you to resize the primitive as required.

To select several primitives, either drag out a selection rectangle around the primitives you want to select, or select each primitive in turn, holding down the **SHIFT** key to indicate that you want each primitive to be added to the selection. If multiple primitives are selected, the

red rectangle will surround all of the primitives, and the handles can then be used to resize the primitives as a group. The relative size and position of the primitives will be maintained, as long as Crimson can do so without violating minimum size requirements.

BURIED PRIMITIVES

If you find that the primitive you want to select is hidden below another primitive, press the **CTRL** key to allow the selection to be made. Alternatively, right-click to access the context menu, and choose the Select submenu. This will list the all primitives that are beneath the mouse pointer, ordering them from back to front. Each command will select the corresponding primitive, making it easy to ensure that you have selected the correct element.

USING THE QUICK BAR

The Quick Bar is a floating toolbar that appears to the top-right of the current selection…

The bar will at first appear in a faded form, and will become more solid as you move your mouse towards it. Moving away from it will hide the bar, after which it will not reappear until the selection process is repeated, or the scroll-wheel button on your mouse is pressed. The Quick Bar allows easy access to a number of commonly-used features while minimizing mouse movement. The bar can be enabled or disabled using a command on the View menu.

MOVING PRIMITIVES BETWEEN PAGES

Primitives can be dragged around a display page in the usual way, but can also be copied from one page to another. To do this, select the primitive you wish to copy and drag it towards the Navigation Pane. If the pane is hidden, hover over the arrowed bar and the pane will slide into view. Hover over the target page, and that page will be selected. Now drag the primitive back into the editor and drop it on the new page. Holding down **CTRL** will change the copy operation to a move, working in an opposite sense as when moving within a page.

MOVING PRIMITIVES BETWEEN DATABASES

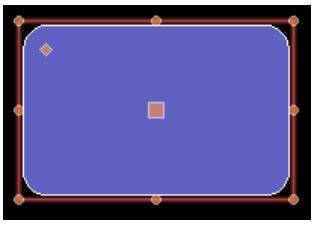
Dragging primitives between databases is just as easy. Simply select the items you wish to copy, and drag them to another copy of Crimson that contains the new database. This will work with entire pages, groups of primitives or just a single item.

CHANGING THE SIZE OF PRIMITIVES

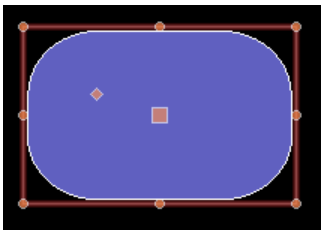
Resizing primitives is performed in the intuitive manner of grabbing one of the sizing handles and moving it in the required direction. The **CTRL** key can be held down to restrict the sizing operation such that the primitive's width and height are equal. The **SHIFT** key can be held down to allow the sizing to operate from the "middle out" rather than from one edge.

USING LAYOUT HANDLES

Certain primitives have internal handles that can be moved to change their layout. For example, the rounded rectangle shown below has a single layout handle in its top left-hand corner. The handle is marked with a diamond whenever the primitive is selected…



In this case, moving the handle changes the radius of the rectangle's corners…

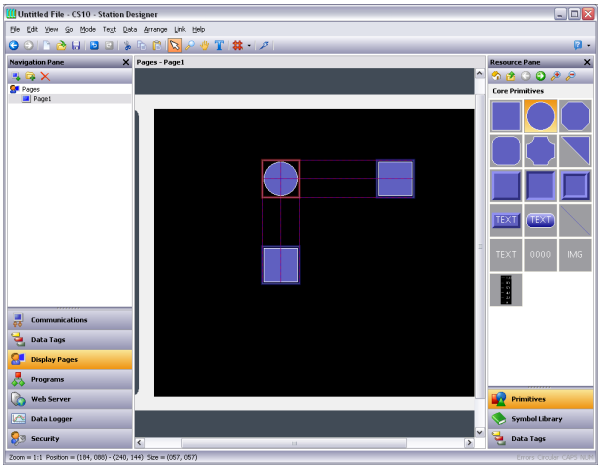


The function of each handle depends on the specific primitive, but is usually intuitive.

SMART ALIGNMENT

If you have the Smart Align features of the View menu enabled, Crimson will provide you with guidelines during a move or size operation. These will help align a primitive with existing primitives, or with the center of the display. With a little practice, this feature can make it very easy to align primitives as they are created, without the need to go back and "tweak" your display pages to get the various figures into alignment.

In the example shown below, a circle is being aligned with two squares…

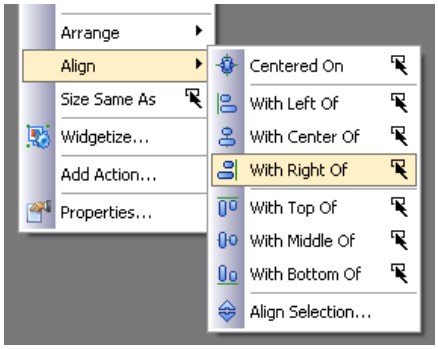


Guidelines are present at both the edges of the figures, and at the center, showing that both the edges and the centers are aligned. The red rectangle is highlighting the primitive that is

being manipulated, while the blue rectangles are highlighting the primitives to which the guidelines have been drawn.

QUICK ALIGNMENT

Crimson's Quick Alignment features allow primitives to be aligned to other primitives without the need to bring up a dialog box. To use this feature, simply select the primitive you want to move, and right-click to bring up the context menu. Select the Align submenu, and then select one of the various "With...Of" options, marked with the rectangle-and-cursor symbol. The mouse pointer will change to indicate that you now need to click on the primitive to which you wish to align.



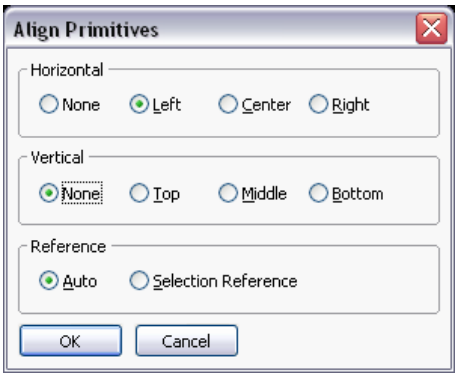
As soon as you click, the alignment will be performed.

USING THE GRID

The Grid button on the toolbar can be used to control the behavior and the display of the alignment grid. Clicking on the left-hand side of the button will show or hide the grid. Clicking on the drop-down portion will allow the operations for which the grid is used to be configured. You may separately enable or disable the grid for creation, sizing and movement operations, or you may use the All or None options to enable or disable it globally. You may also control whether the grid is used when editing within groups.

ALIGNING PRIMITIVES

While the Smart Alignment and Quick Alignment options discussed above allow many alignment operations to be performed without further concern, there are times that you will want to use a more traditional approach. To do this, select a number of primitives, and use the Align Selection command on the Arrange menu to display the following dialog box…



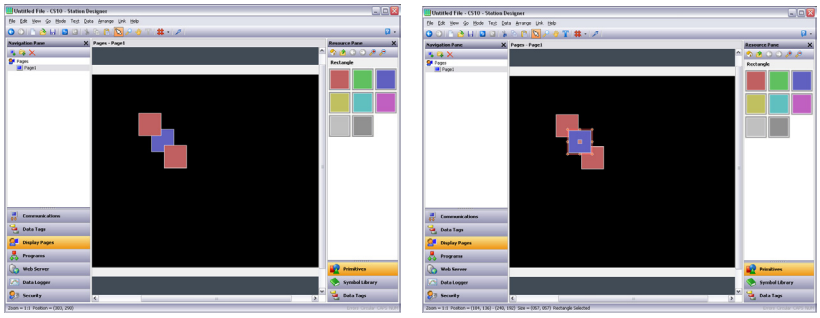
The Horizontal and Vertical settings can be used to indicate what type of alignment is to be performed, while the Reference setting defines which primitive is used as the reference for the alignment operation. In the example above, Auto mode will use the left-most primitive as a reference as we are performing a left alignment. Other alignment modes work in a similar way. The alternative mode uses the first selected item as a reference. This item can be identified by the larger square at its center.

SPACING PRIMITIVES

If you have a number of primitives that you wish to space equally on the page, you may use the Space Equally Vertical or Space Equally Horizontal commands on the Arrange menu. The commands work on the currently selected primitives, and attempt to reallocate the free space between the items to achieve equal spacing. The two outer primitives will be left in their current positions. Note that the command may fail if an inappropriate set of primitives are selected, and may not achieve perfect spacing if the available space is too limited.

REORDERING PRIMITIVES

Primitives on a display page are stored in what is known as a z-order. This defines the sequence in which the primitives are drawn, and therefore whether or not a given primitive appears to be in front of or behind another primitive. In the first example below, the blue square is shown behind the red squares i.e. at the bottom of the z-order. In the second example, it has been moved to the front of the order, and appears in front of the other figures.



To move items in the z-order, select the items, and then use the various commands on the Arrange menu. The Move Forward and Move Backward commands move the selection one step in the indicated direction, while the Move To Front and Move To Back commands move the selection to the indicated end of the z-order. Alternatively, if you have a mouse that is equipped with a wheel, the wheel can be used to move the selection by moving the wheel with the **CTRL** key held down. Scrolling up moves the selection to the back of the z-order; scrolling down moves the selection to the front.

DUPLICATING PRIMITIVES

The **CTRL+D** key combination or the Smart Duplicate command on the Edit menu can be used to make a copy of the current primitive, adjusting its properties such that the primitive gets it controlling data from the next data item. The definition of "next" depends on the exact type of the data, with Crimson being capable of selecting the next register in a comms device, the next member of an array, or the next tag in a sequence. As an example, repeatedly using Smart Duplicate with a button mapped to `Array[0]` will produce a sequence of buttons mapped to `Array[1]`, `Array[2]` and so on until the whole screen is filled.

EDITING MULTIPLE PRIMITIVES

You may on occasions want to edit the properties of multiple primitives. Crimson supports this by having you edit one primitive, and then allowing you to set the properties of a number of other primitives equal to those of the one that you first edited. Crimson provides two methods to do this, both of which rely on the same underlying mechanism.

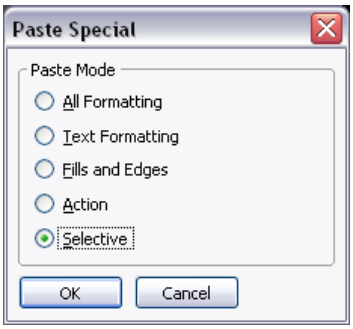
USING COPY FROM

The Copy From command can be used to copy the selected properties of a given primitive to one or more other primitives. To use the command, select the required targets, and then right-click to access the associated context menu. Select one of the Copy From commands, and the cursor will change to allow you to select the primitive from which the copy operation should be performed. Depending on the command that was selected, one or more properties from the source will then be applied to the target primitives.

USING PASTE SPECIAL

The Paste Special command can be used to achieve the same result, but via a different method that also allows properties to be copied between databases and between multiple instances of Crimson. First, select the source primitive and use the Copy command to put it on the

Clipboard. Then, select the required target primitives, right-click the selection, and select the Paste Special command. The following dialog box will appear…



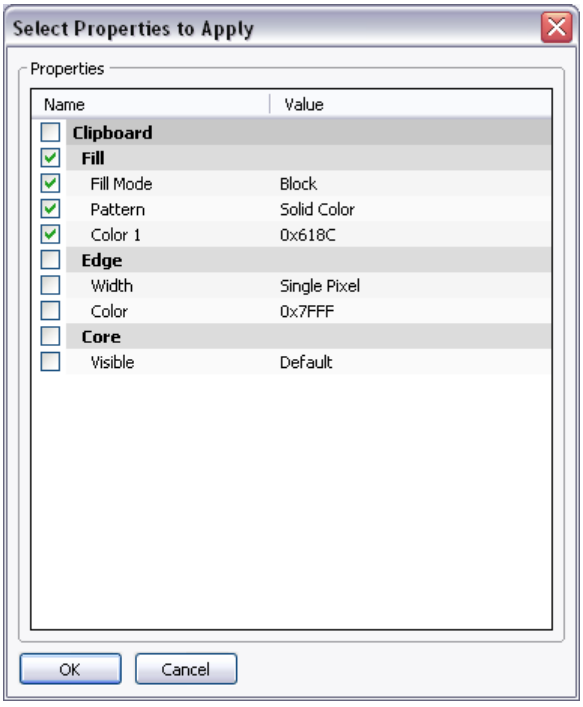
The selected properties from the source primitive will be applied to the target primitives.

PROPERTY SELECTIONS

Both methods detailed above allow you to define which properties are to be copied…

- *All Formatting* copies everything except any text, data item or action.
- *Text Formatting* copies the font, alignment and margins of text or data items.
- *Fills and Edges* copies the fill and edge attributes from the Figure tab.
- *Action* copies any action assigned to the primitive.

In addition, the *Selective* option can be used to select the properties to copy…

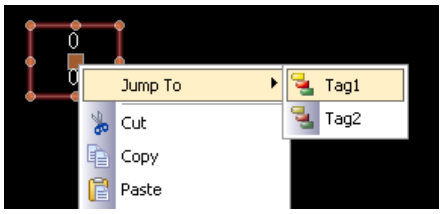


The list contains a hierarchical presentation of the properties defined by the source primitive, organizing them according to the layout used when editing the primitive, and showing the

value assigned to each. Each property or group of properties can be selected or deselected using the associated checkboxes. The checked properties will be applied, thereby providing you with low-level control of what gets copied from one primitive to another.

JUMPING TO OTHER ITEMS

If a primitive references tags, display pages or other items, a Jump submenu will appear on its context menu. Select this menu to view a list of referenced items. Select one of those items to jump directly to that section of the database. The example below shows a primitive that references two tags…



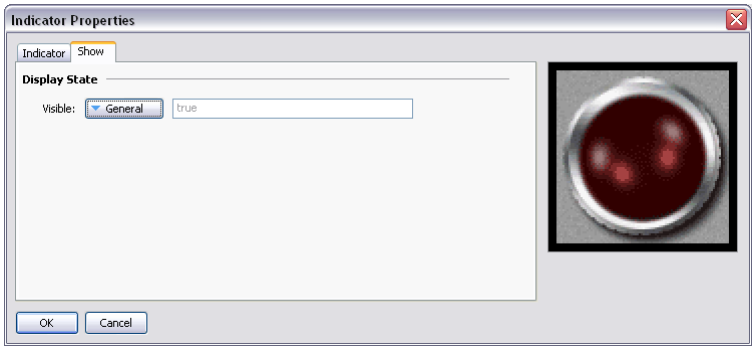
After you have made whatever changes you want to the tag, you can use the Back button on the toolbar or the **ALT+LEFT** key combination to return to the display page that you were just editing. Note how the selection is preserved during navigation, making it easy to view or edit a referenced object and to then resume the display creation process.

PRIMITIVE PROPERTIES

The properties of a primitive can be edited by double-clicking on the primitive, or by using the Properties command on the primitive's context menu. You may also select the primitive and press the **ALT+ENTER** key combination. The property dialog for a primitive will contain various tabs, with some tabs only appearing when additional items—such as text, data or an action—have been added to the primitive. The properties dialog shows a live preview of the current primitive, allowing you to see the effect of changes before you commit them.

SHOWING OR HIDING PRIMITIVES

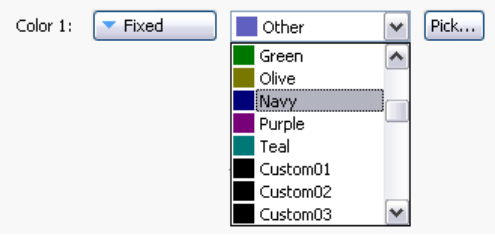
All primitives have a Show tab in their property dialog…



The *Visible* property can be set to an integer expression to show or hide the associated primitive at runtime. A value of zero will hide the primitive, while a non-zero value will allow it to be shown. All primitives are visible by default.

DEFINING PRIMITIVE COLORS

Colors within primitives are edited using a field similar to that shown below…



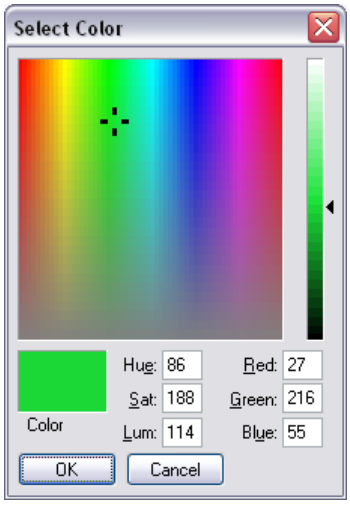
You will note that the color property is presented by means of a drop-down menu button, a drop-down list and a Pick button. The drop-down menu selects the color animation mode, which can be any one of the following…

- In *Fixed* mode, the color does not change, and is selected from the drop-down list, or by invoking the color selection dialog by pressing the Pick button.
- In *Tag Text* mode, the color is animated to match the foreground color defined by a particular tag. The specific tag can be selected by pressing the Pick button.
- In *Tag Back* mode, the color is animated to match the background color defined by a particular tag. The specific tag can be selected by pressing the Pick button.
- In *Flashing* mode, the color is animated to alternate between two colors at a specific rate, with another color being displayed when flashing is disabled.
- In *2-State* mode, the color is animated to switch between two colors depending on the value of a tag or other data item.
- In *4-State* mode, the color is animated to switch between four colors depending on the value of two tags or other data items.
- In *Blended* mode, the color is animated to transition smoothly from one color to another based upon the value of a tag or other data item relative to specified minimum and maximum values.
- In *Expression* mode, a numeric expression can be entered that will be used to determine the color to be displayed. See below for more details.
- In *Complex* mode, a local program returning an integer value can be written to define the color to be displayed. See below for more details.

The drop-down menu contains the following colors…

- The sixteen standard VGA colors.
- Thirty-two shades of gray between black and white.
- Any other colors used in the database, up to a limit of twenty-four.

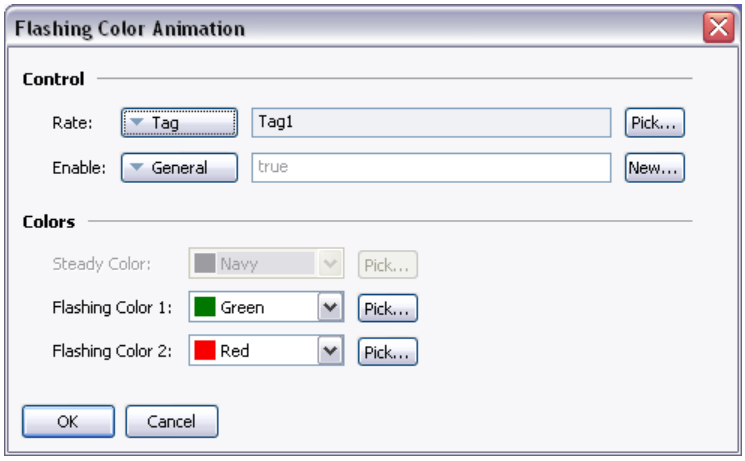
The More option at the bottom of the list can be used to invoke the color selection dialog…



This dialog offers several ways of defining a color. You can pick from the palette, pick from the "rainbow" window, or enter the explicit HSL or RGB parameters. If the color selected has not previously been used in the database and is not one of the standard colors or grays, it will be added to the custom colors shown in the drop-down menu.

DEFINING FLASHING COLORS

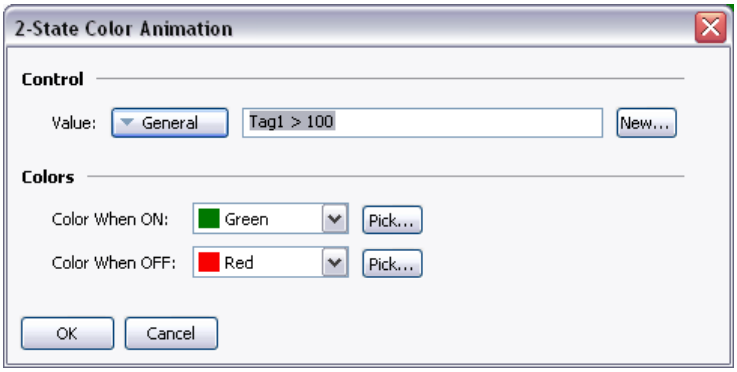
Flashing colors are defined via the following dialog box…



- The *Rate* property defines the rate at which the flashing should occur. A value of 1 produces a flashing rate of 1Hz, with each color being displayed for 500ms. It is not recommended to use rates in excess of 4Hz, as the target device's display update rate may produce unpleasant "beating" effects.
- The *Enable* property defines an optional expression that can be used to enable or disable flashing. The Steady Color will be displayed when flashing is disabled.
- The *Color* properties allow you to define the colors to be used.

DEFINING 2-STATE COLORS

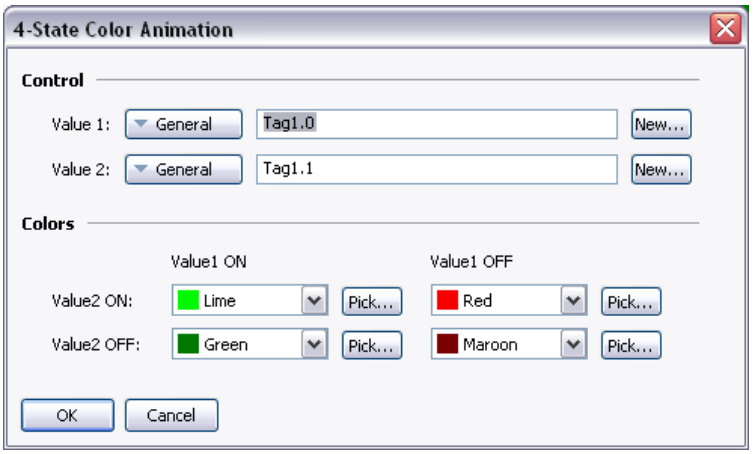
2-State colors are defined via the following dialog box…



- The *Value* property is used to select the color to be displayed.
- The *Color* properties allow you to define the colors to be used.

DEFINING 4-STATE COLORS

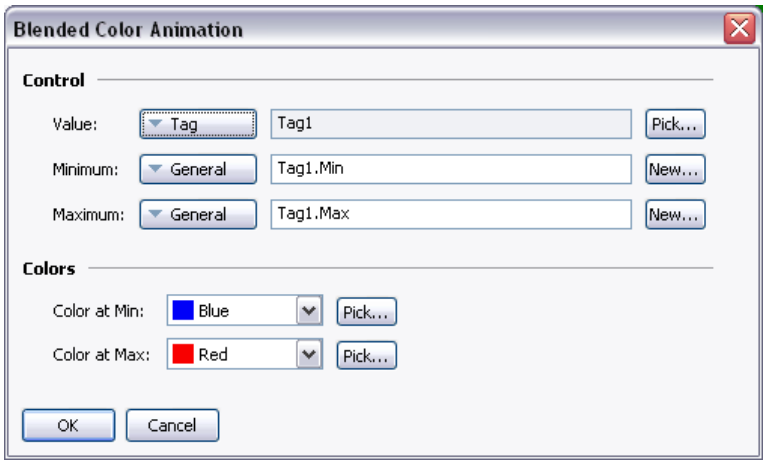
4-State colors are defined via the following dialog box…



- The *Value* properties are used to select the color to be displayed.
- The *Color* properties allow you to define the colors to be used.

DEFINING BLENDED COLORS

Blended colors are defined via the following dialog box…



- The *Value, Minimum* and *Maximum* properties are used to define the color to be displayed. In the example shown, the color will blue when the tag is at or below its minimum value, red when it is at or above its maximum value, and will transition smoothly from blue to red as the tag changes between its limits.
- The *Color* properties allow you to define the colors to be used.

DEFINING COLOR EXPRESSIONS

As mentioned above, color properties can be defined via integer expressions or via local programs returning integer values. These mechanisms are used in those circumstances where the standard color animation methods are not sufficient. As you will rarely have to use these features, feel free to skip this section if you consider it too complex.

Crimson works with 15-bit color values, with the lowest five bits representing the red, the next five bits representing the green and the upper five bits representing the blue. You can manipulate color values just as you would any other integer value.

BUILDING COLORS

The `ColGetRGB(r,g,b)` function can be used to create a color value from its red, green and blue components. Although Crimson uses 15-bit color values containing three 5-bit values, the arguments passed to this function are scaled down by a factor of 8 and should thus be in the range 0 to 255. `ColGetRGB(128, 0, 64)` will thus return a purple-like color with a red value of 16, no green component and a blue value of 8.

SPLITTING COLORS

The `ColGetRed(rgb)`, `ColGetGreen(rgb)` and `ColGetBlue(rgb)` functions can be used to access the individual color components of a color value. In keeping with the convention used by `ColGetRGB()`, the values returned by these functions are scaled to be between 0 and 255.

CHOOSING COLORS

The `ColPick2()` function can be used to select between two colors based on the value of an expression. For example, the expression `ColPick2(Flag1, Col1, Col2)` will return `Col1`

if `Flag1` is non-zero, or `Col2` if `Flag1` is zero. The first and second color arguments can be replaced by calls to the `ColGetRGB()` function if required.

BLENDING COLORS

The `ColBlend()` function can be used to produce a color that is a user-defined blend of two other colors. For example, the expression `ColBlend(Data, 0, 100, Col1, Col2)` will return `Col1` if `Data` is 0 and `Col2` if Data is 100. Intermediate values will be appropriate mixtures of the two colors, allowing a smooth transition from one color to another. Once again, the color arguments can be replaced by calls to the `ColGetRGB()` function.

RESPONDING TO TOUCH

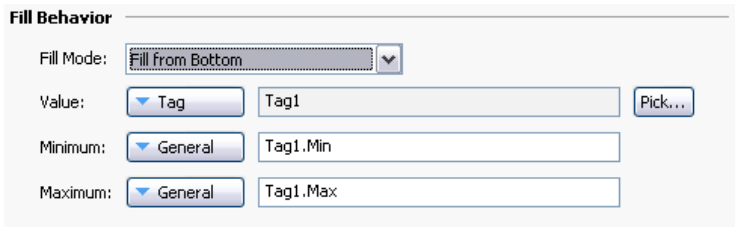
The `IsPressed` system variable is equal to true if the current primitive has been touched and false otherwise. It can be used with the color selection functions to animate a primitive according to its touch status. Note that primitives will not be enabled for touch unless they have an action defined or they support an inherent action.

DEFINING TANK FILLS

Many geometric primitives support a so-called "tank fill" option whereby the figure is filled to a given level based upon the contents of a tag. This feature can be used to implement simple bar graphs, or to fill more complex shapes.

The example below shows a six-pointed star with a bottom-up tank fill set to 60%...

Tank fills are defined using a primitive's Fill Behavior properties…

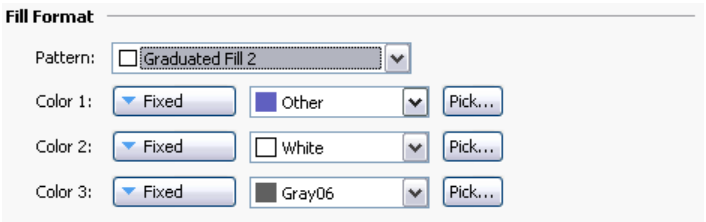


- The *Fill Mode* property defines whether a tank fill should be drawn, and from which direction the fill should occur. Fills can occur from any edge of the primitive, allowing complex animations to be created. Block mode results in the figure being filled with a single pattern, disabling tank fills.
- The *Value* property selects the value used to calculate the level of the fill. If a tag is entered, the Minimum and Maximum limits will automatically be set to the data entry limits of that tag using the tag property expression syntax. The Value property may be an integer or a floating-point value. The fill level calculations are always performed in floating point.

- The *Minimum* and *Maximum* values define the limits to be used when scaling the Value property to calculate the fill level.

DEFINING FILL FORMATS

A primitive's Fill Format properties define how the inside of the primitive will be filled…



- The *Pattern* property selects between various fill patterns. The usual option is Solid Color, but a variety of dithered and hatched patterns may also be selected. A number of graduated fills are also available…

PATTERN	DESCRIPTION
Graduated Fill 1	Color 1 at the top and bottom of the primitive, changing vertically to Color 2 at the center.
Graduated Fill 2	Color 1 at the top of the primitive, changing vertically to Color 2 at the bottom.
Graduated Fill 3	Color 1 at the left and right of the primitive, changing horizontally to Color 2 at the middle.
Graduated Fill 4	Color 1 at the left of the primitive, changing horizontally to Color 2 at the right.

- The *Color 1* property defines the first color to be used for the fill.
- The *Color 2* property defines an optional second color to be used for the fill.
- The *Color 3* property defines the background color for a tank fill. It is not required if a block fill is being used. The property may not be present if the current primitive does not support tank fills.

DEFINING EDGE FORMATS

A primitive's Edge Format properties define how the edge of the primitive will be drawn…

- The *Width* property specifies the thickness of the edge. The edge may be displayed by selecting a value of None. Crimson currently supports only odd edge sizes, up to nine pixels in width.
- The *Color* property defines the color of the edge.

- The *Corners* property is only present for rectangles, and defines whether rounded or square corners should be used when drawing the edge. All other primitives use rounded corners by default.

USING GROUPS

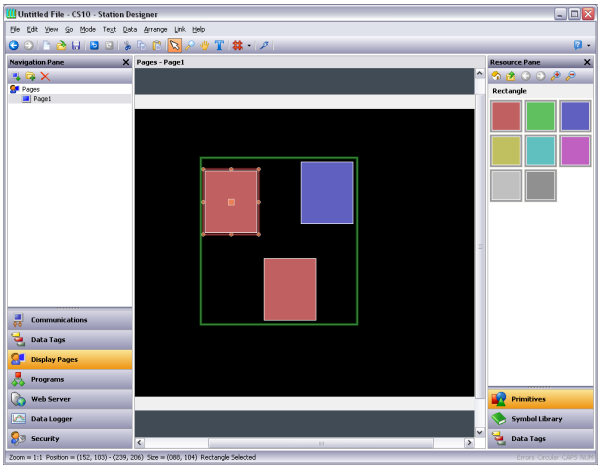
A group is a collection of primitives that is treated as a single object.

MAKING AND BREAKING GROUPS

If you have several primitives that you wish to treat in this way, you may select them as described above and then use the Group command on the Organize menu. You can perform the same operation by pressing the **CTRL+G** key combination. Once a group has been created, it can be moved, sized and copied just like a single object. A group can be broken into its component primitives by selecting it and using the Ungroup command, or the **CTRL+U** key combination. Note that groups can comprise both primitives and other groups, and that groups can be nested up to any reasonable limit.

EDITING WITHIN GROUPS

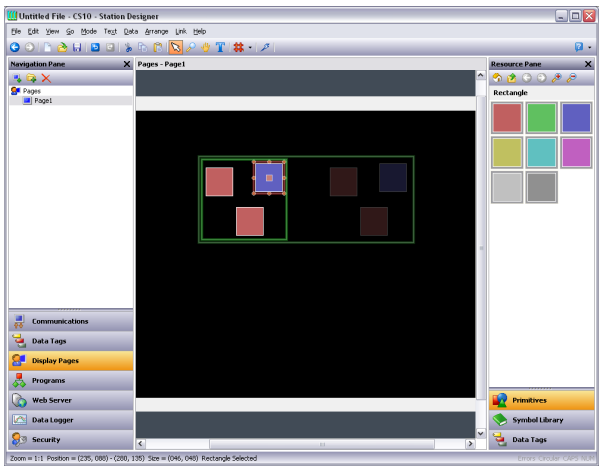
Once a group has been created, you might want to edit its contents without first breaking it apart. This is particularly useful when you have created nested groups, as the regrouping process would then be very difficult. To edit within a group, first select that group, and then click on a member of the group. (Avoid clicking on the central handle of the group object, as that is used to move or select the group as a whole.) Once the group member has been selected, Crimson will switch into group editing mode, as shown below…



Note the green rectangle displayed around the group that is being edited. Editing within a group works just like editing within a page, except that items cannot be moved beyond the group boundaries. They can be copied, pasted, sized, and deleted. In fact, any of the usual operations can be performed. You can even drag new items from the Resource Pane and drop them into a group. To exit group-editing mode, click outside the group or press the **ESC** key.

NESTED GROUP EDITING

Crimson also allows editing within groups that are themselves within groups…



To activate this feature, begin editing within the outer group, select the inner group and then click on a member of that inner group. Note in the example above how a series of fading rectangles are used to show the group hierarchy. Note also how items outside the current groups are shown in faded colors to make it easier to see where the group ends. When using the **Esc** key to exit nested group editing, each press of the key will move up one level.

EXPANDING GROUPS

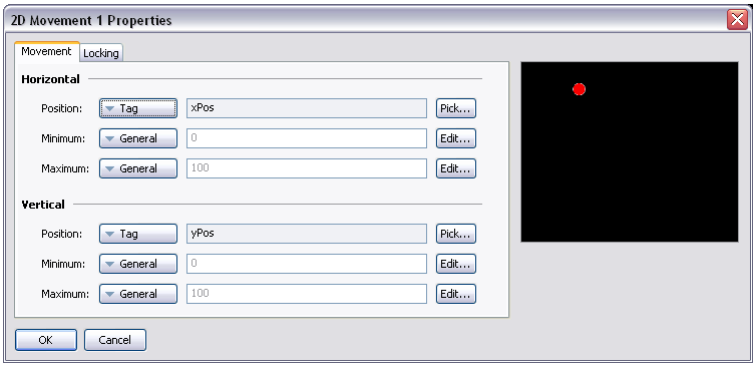
As mentioned above, movement of primitives during group editing is limited such that you cannot move a primitive outside the group to which it belongs. In situations where you want to make adjustments to primitives at the edge of a group, you may select the Expand and Edit command from context menu of the group. This will move the group boundaries out from the primitives that it contains, allowing such adjustments to be made. When group editing mode is cancelled, the group boundary will be moved inwards to tightly surround its contents.

ADDING MOVEMENT TO PRIMITIVES

Any primitive can be animated such that it moves dynamically within a bounding rectangle that you define. Primitives can be moved horizontally, vertically or in both dimensions, or they can be moved according to polar coordinates such that they orbit a point at a variable distance. In each case, each dimension is defined by a control value and a pair of limits.

To apply movement, select the required primitive or primitives, and choose one of the Add Movement commands from the Behavior menu. A red rectangle will appear around the primitives, representing the movement group in which the animation will take place. The group can be resized to change the extent of animation, but unlike a regular group, resizing a movement group does not change the size of the primitives themselves. The group contents can be edited just as with standard groups, using the techniques detailed above. When polar movement is being edited, an ellipse will show the path that the primitives will follow when the radius is set to 100%. Note that this will always be smaller than the group itself, as it represents the position of the center of the items that are being animated, and space must be left to ensure that they do not extend beyond the group boundary.

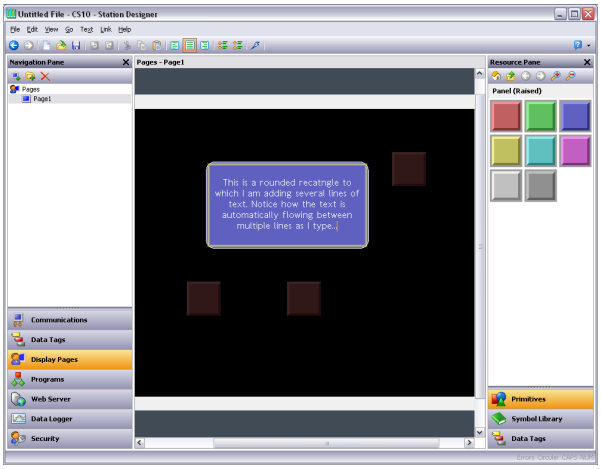
To define how the movement is controlled, open the properties of the movement group…



The example above shows the configuration of 2D movement. Polar movement is configured in a similar way. For each dimension of movement, the *Position* value defines where the contents of the group will be placed relative to its outline. The *Minimum* and *Maximum* values represent the limits of the control values. For 2D movement, the minimum settings resulting the group contents being in the top left corner.

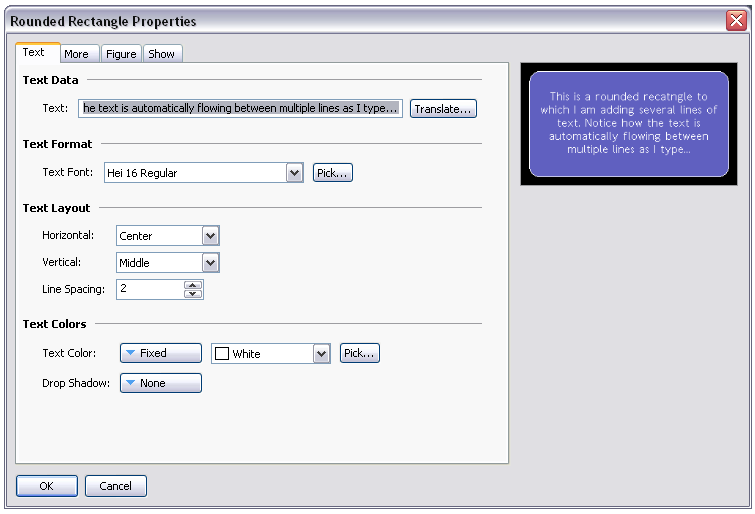
ADDING TEXT TO PRIMITIVES

Most primitives within Crimson can support the addition of text. To add text to a primitive, simply select the primitive, press **F2** and begin typing. Alternatively, you can right-click the primitive and select the Add Text command from the resulting menu. The example below shows text being entered into a rounded rectangle…



Note first of all how the bounding rectangle for the primitive is shown in yellow, and how all the other primitives on the page are faded out. Note also how the text editor automatically splits the text across lines. Try resizing a primitive containing text, and you will see how Crimson reflows the text to fit into the new shape.

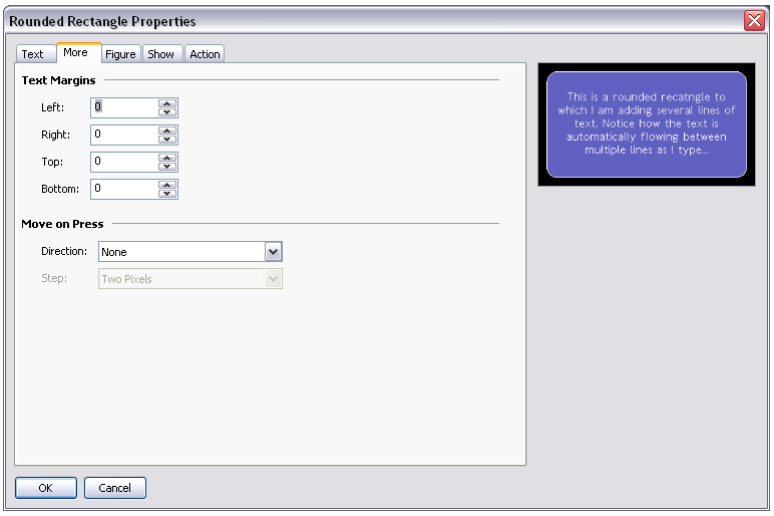
During text editing, the toolbar changes to provide commands to modify the text alignment, and to grow or shrink the spacing between lines. The more advanced text properties can be editing by either selecting Text Properties from a primitive's context menu, or by pressing **ALT+ENTER** while in text editing mode…



TEXT PROPERTIES

- The *Text* property contains the text to be displayed. Vertical bar characters are used to encode hard line breaks. Since this field is a translatable string, multilingual versions can be edited. This also implies that the property can be set to an expression, allowing its contents to change dynamically. Crimson supports full dynamic reflow, allowing complex and attractive presentation options.
- The *Text Font* property allows the required font to be selected. Crimson's new default font is Hei—a Unicode font that provides support for simplified Chinese and most other languages. The Pick button can be used to invoke the font selection dialog, allowing any font that is installed on your system to be rendered in a form that can be used by the target device. Note that it is your responsibility to ensure that you are licensed for this kind of font usage.
- The *Horizontal* property defines the horizontal alignment of the text.
- The *Vertical* property defines the vertical alignment of the text.
- The *Line Spacing* property defines additional line spacing in pixels.
- The *Text Color* property selects the color of the text.
- The *Drop Shadow* property is used enable an optional shadow to the right and to the bottom of the text itself. This effect is useful when trying to make text stand out from its background, especially if the background is an image that contains a combination of many colors.

MORE PROPERTIES

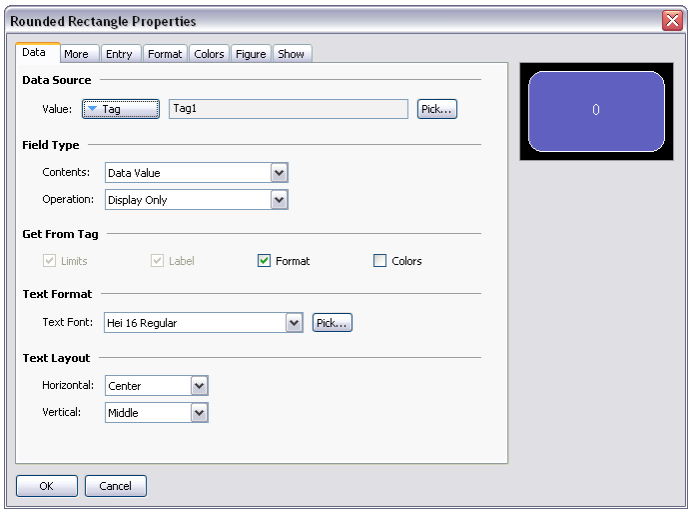


- The *Text Margin* properties are used to control the margin around the text relative to the text-bounding box provided by the primitive. They can be useful in achieving better visual centering when working with fonts that have lots of space above or below their characters, either for diacriticals or descenders.
- The *Direction* property defines the direction in which the text will be moved when the associated primitive is pressed. It is only enabled when an action is assigned to the primitive, or when the primitive is something like a button that has an inherent action associated with it. This option is useful when creating custom buttons that should provide feedback when touched.
- The *Step* property indicates how far the text should move when the primitive is pressed. One to three pixels can be chosen, according to the effect desired.

ADDING DATA TO PRIMITIVES

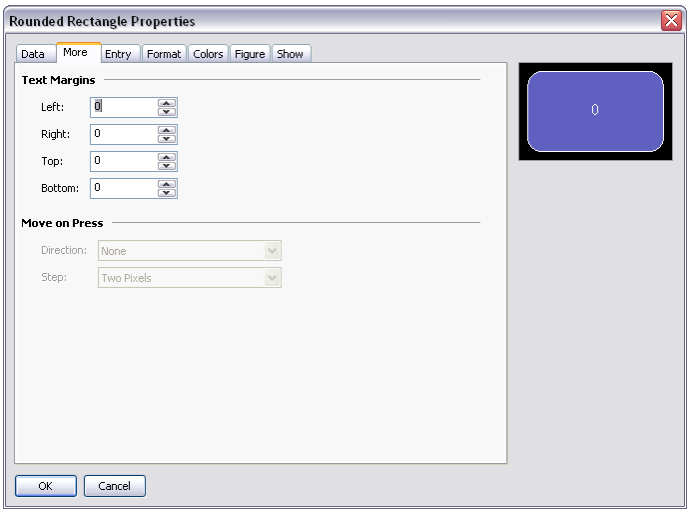
Primitives which support the addition of text also support the display of live data, and can optionally be configured for data entry. To add data to a primitive, right-click the primitive and select the Add Data command from the resulting menu. Alternatively, select the primitive and press the **CTRL+F2** key combination. The primitive's properties dialog will be displayed, with a number of additional tabs being available to define the data item and its behavior.

DATA PROPERTIES



- The *Value* property defines the data value to be displayed.
- The *Contents* property defines whether the field should display the data value, the data value and its associated label, or just the label alone.
- The *Operation* property defines whether the field should just display the value or also provide data entry functionality. Data entry is obviously only available if the selected data value is writable.
- The *Get from Tag* properties define whether certain properties of the data field are defined locally or are linked to the properties of the tag being displayed. The options are only available when a tag is specified in Value.
- The *Text Font* property allows the required font to be selected. Crimson's new default font is Hei—a Unicode font that provides support for simplified Chinese and most other languages. The Pick button can be used to invoke the font selection dialog, allowing any font that is installed on your system to be rendered in a form that can be used by the target device. Note that it is your responsibility to ensure that you are licensed for this kind of font usage.
- The *Horizontal* property defines the horizontal alignment of the text.
- The *Vertical* property defines the vertical alignment of the text.

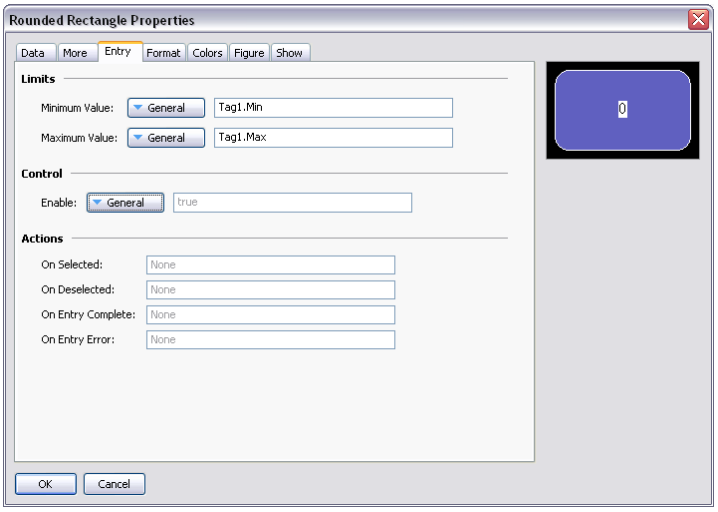
MORE PROPERTIES



- The *Text Margin* properties are used to control the margin around the text relative to the text-bounding box provided by the primitive. They can be useful in achieving better visual centering when working with fonts that have lots of space above or below their characters, either for diacriticals or descenders.
- The *Direction* property defines the direction in which the text will be moved when the associated primitive is pressed. It is only enabled when an action is assigned to the primitive, or when the primitive is something like a button that has an inherent action associated with it. This option is useful when creating custom buttons that should provide feedback when touched.
- The *Step* property indicates how far the text should move when the primitive is pressed. One to three pixels can be chosen, according to the effect desired.

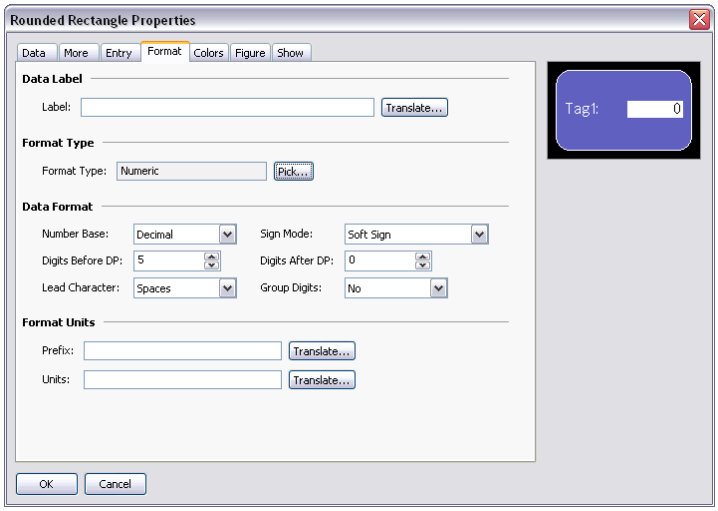
ENTRY PROPERTIES

These properties are only available when data entry is enabled…



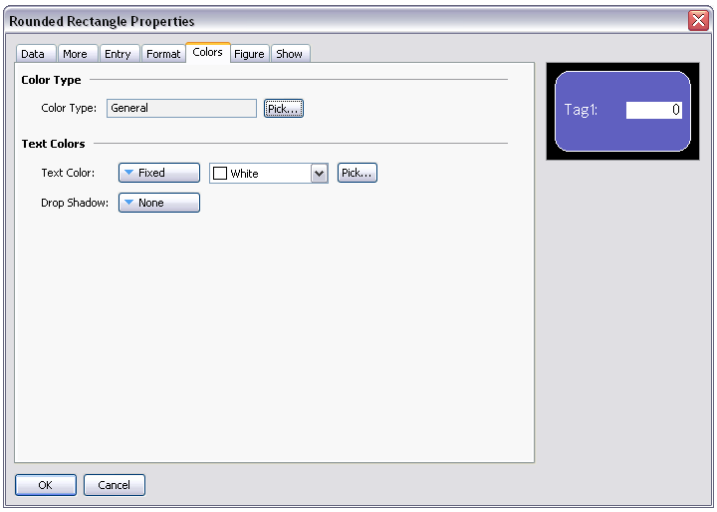
- The *Maximum Value* and *Minimum Values* properties define the data entry limits. They will not be available if the field has been configured to get its data entry limits from the controlling tag. Not all format types honor these settings, particularly if their data limits are implicitly defined.
- The *Enable* property is used to provide an expression to enable or disable data entry. Disabled data entry fields will act just like display-only fields.
- The *On Selected* property specifies an action to be executed when the user presses on the data entry field, just before data entry begins.
- The *On Deselected* property specifies an action to be executed when data entry ends, either as a result of a value being written, a page change or the user pressing a button to cancel the entry process.
- The *On Entry Complete* property specifies an action to be executed when data entry is successfully completed.
- The *On Entry Error* property specifies an action to be executed when the user enters an invalid value.

FORMAT PROPERTIES



- The *Label* property defines the label to be applied to this field. It may not be available if the label is not to be displayed, or if the field is configured to get its label from the controlling tag.
- The *Format Type* field specifies the format type to be used when displaying and optionally editing the data value. Again, the selection may not be available if the format is being obtained from the controlling tag.
- Other properties are specific to the data format that has been selected. Refer to the chapter on Using Formats for details of each format's properties.

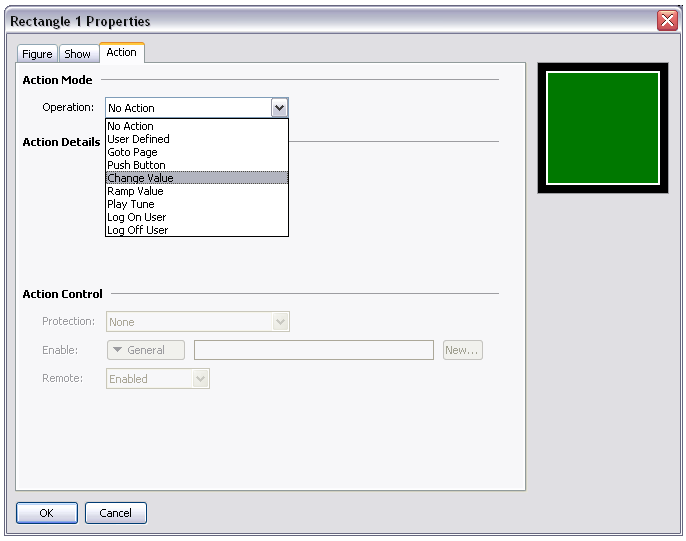
COLOR PROPERTIES



- The *Color Type* field specifies the coloring to be used when displaying the data value. The selection may not be available if the coloring is being obtained from the controlling tag.
- The *Text Color* property is used to override the color of the text if the General coloring is being used.
- The *Drop Shadow* property is used to enable an optional shadow to the right and to the bottom of the text itself. This effect is useful when trying to make text stand out from its background, especially if the background is an image that contains a combination of many colors. It is only available with the General coloring.
- Other properties are specific to the coloring that has been selected. Refer to the chapter on Using Colorings for details of each coloring's properties.

ADDING ACTIONS TO PRIMITIVES

Primitives that do not perform their own implicit action support the addition of customized actions to be performed when the operator presses or releases the touch-screen. An action can be added by selecting the Add Action command from the primitive's context menu, or by selecting the primitive and pressing the **CTRL+I** key combination. An Action tab will be added to the primitive's properties, and the properties dialog will appear…



PROTECTING ACTIONS

An action's Protection property can be used to prevent an action from being invoked accidentally. This facility operates in addition to any protection provided by the Security System, and is invoked before the associated actions are begun. The following protection modes are available…

- *Confirmed* mode displays a popup to confirm the action, and then performs the action immediately if the user indicates that the action should proceed.
- *Locked* mode displays a popup stating that the action is locked. If the user indicates that the action should proceed, it becomes unlocked, and they must activate the action again for it to actually take place. Selecting another action will lock the previous action, as will waiting beyond the global timeout.
- *Hard Locked* mode operates as for Locked mode, except that the action will relock once it has been performed and must be unlocked each time.

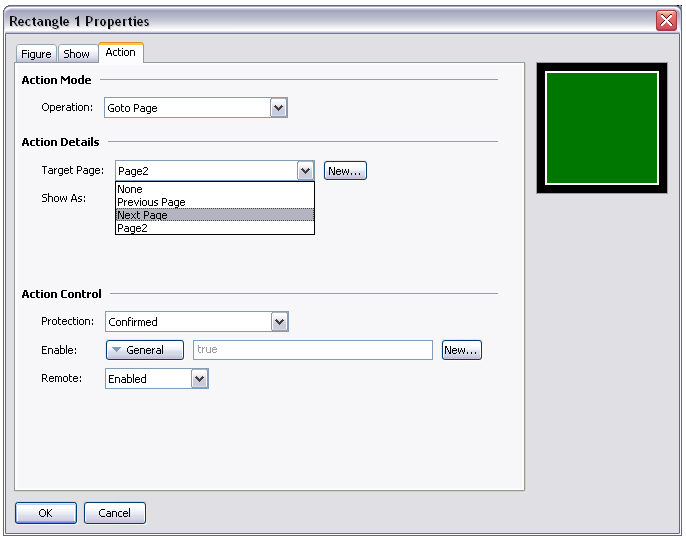
ENABLING ACTIONS

If you want to make a particular action dependent on some condition being true, enter an expression for that condition in the *Enable* field. This expression may reference a flag tag directly, or may use any of the comparison or logical operators defined in the Writing Expressions section. If you need more complex logic such that one of several actions is performed based on more complex decision-making, configure the key for User Defined

mode, and use it to invoke a program that implements the required logic. You can also use the *Remote* property to block access to this action from the web server.

THE GOTO PAGE ACTION

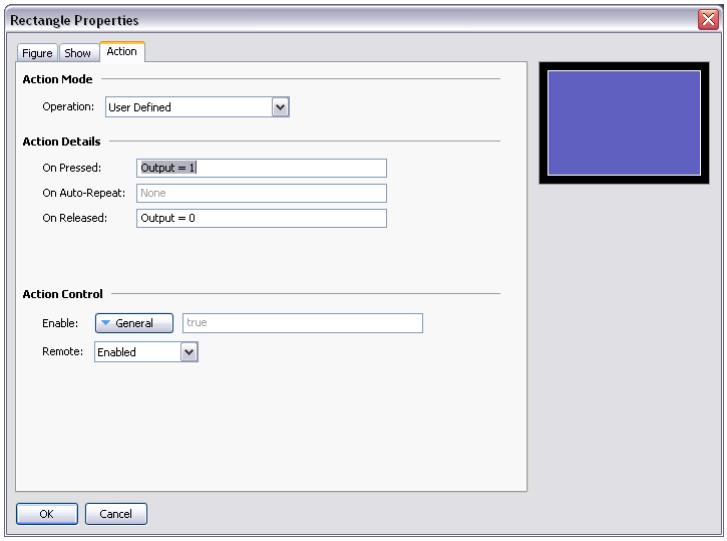
This action is used to instruct the target device to show a new page…



- The *Target Page* property is used to indicate which page should be displayed. In addition to the pages contained in the database, you have the option of selecting either Previous Page or Next Page to navigate within the page history list. The New button may be used to create a new page without leaving the dialog.
- The *Show As* property is used to indicate how the page should be displayed. A selection of Normal Page will cause the page to be selected in the usual manner, while the Popup Window option will cause the primitives on the new page to be displayed in a rectangular popup on top of the current page. A popup can be closed by executing the `HidePopup()` function.

THE USER DEFINED ACTION

This action is used to perform one or more user-defined actions…

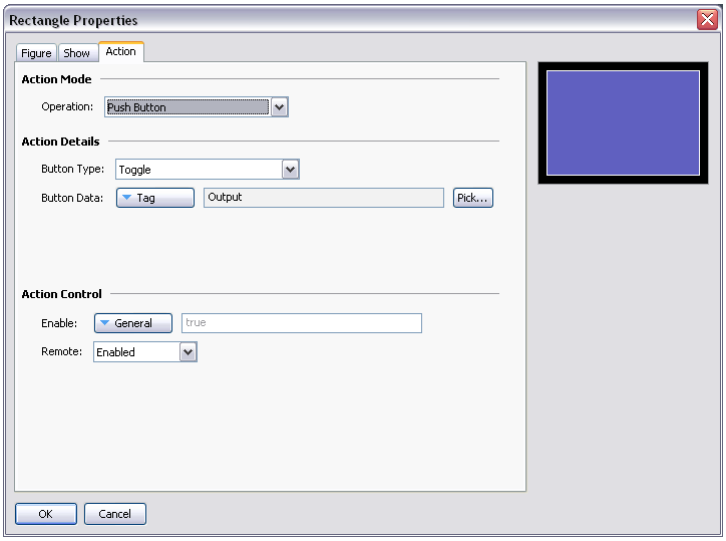


- The *On Pressed* property defines the action to be performed when the primitive is pressed. This action may invoke any of the functions in the Function Reference or the data modification operators described in the Writing Actions chapter. It may also run a program to perform a more complex action.
- The *On Auto-Repeat* property defines the action to be performed when the primitive is pressed and then held down. The action occurs both on the initial depression and on subsequent auto-repeats, so there is no need to define both this property and On Pressed. This action may invoke any of the functions from the Function Reference or the data modification operators described in the Writing Actions section, or it may run a program.
- The *On Released* property defines the action to be performed when the primitive is released. This action may invoke any of the functions from the Function Reference or the data modification operators described in the Writing Actions section, or it may run a program.

In the example above, a user-defined action is used to implement a momentary pushbutton.

THE PUSH BUTTON ACTION

This action is used to emulate a pushbutton…



- The *Button Type* property selects the desired behavior…

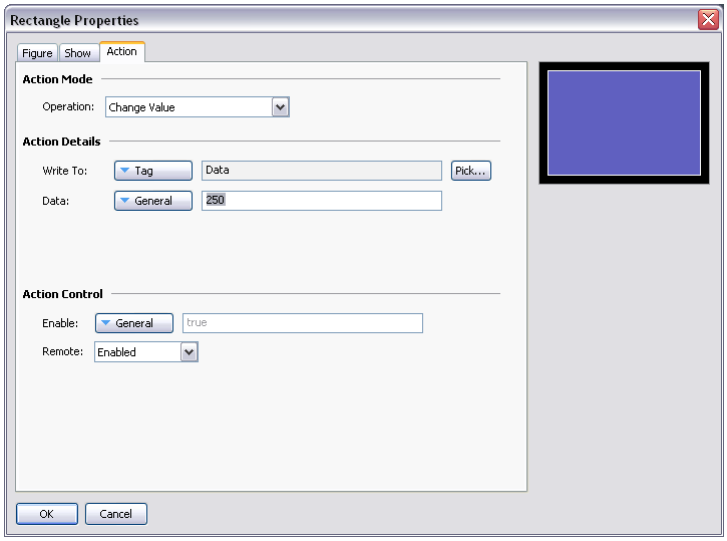
BUTTON TYPE	PRIMITIVE BEHAVIOR
Toggle	Change the data state when the primitive is pressed.
NO Momentary	Set the data to 1 when the primitive is pressed. Set the data to 0 when the primitive is released.
NC Momentary	Set the data to 0 when the primitive is pressed. Set the data to 1 when the primitive is released.
Turn On	Set the data to 1 when the primitive is pressed.
Turn Off	Set the data to 0 when the primitive is pressed.

- The *Button Data* property defines the data to be changed by the key.

  In the example above, touching the primitive will toggle the value of the `Output` tag.

THE CHANGE VALUE ACTION

This action is used to write a numeric value to a data item…

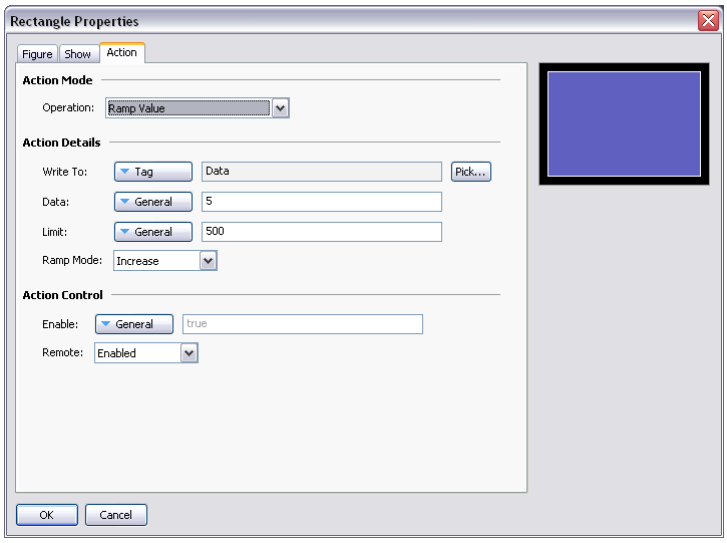


- The *Write To* property defines the data item to be changed.
- The *Data* property defines the data to be written.

In the example above, touching the primitive will set the `Data` tag to 250. Note that this action supports either floating point or integer values. The Data property must be of a type appropriate for the data item defined by the Write To property.

THE RAMP VALUE ACTION

This action is used to increase or decrease a data item. The options are shown below…



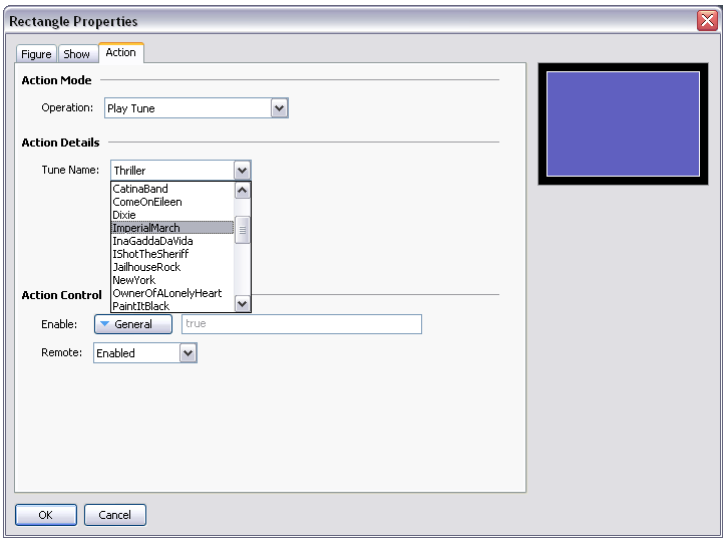
- The *Write To* property defines the data item to be changed.
- The *Data* property defines the step by which to raise or lower the item.

- The *Limit* property defines the minimum or maximum data value.
- The *Ramp Mode* property defines whether to raise or lower the item.

In the example above, pressing and holding down the primitive will repeatedly increase the `Data` tag by 5 until it reaches 500. Note that this action supports either floating point or integer values. The Data and Limit properties must be of a type appropriate for the data item defined by Write To property.

THE PLAY TUNE ACTION

This action plays a selected tune using the target device's internal sounder.



- *Tune Name* selects the tune to be played.

Customized tunes may be played using the `PlayRTTTL()` function.

THE LOG ON USER ACTION

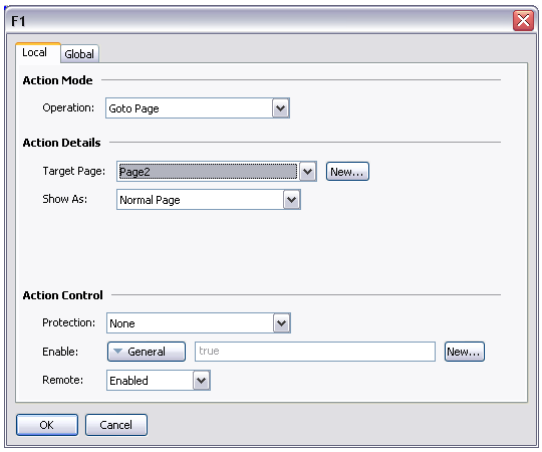
This action activates the log-on screen on the target device. It has no options.

THE LOG OFF USER ACTION

This action logs off the current user of the target device. It has no options.

ADDING ACTIONS TO KEYS

Actions may also be added to the keys of the target device. Zoom out until you can see the keys, and then double-click a key to bring up its properties…



You will notice that this dialog contains two tabs, both of which define an action. The first tab defines the action that will be performed by this key when the current page is displayed, while the second tab defines an action to be performed on every page. These are known as the local and global actions, respectively.

The color used to display the key will change according to which actions are defined…

If the key is displayed in PURPLE, a local action is defined for this PAGE.

If the key is displayed in GREEN, a GLOBAL action is defined.

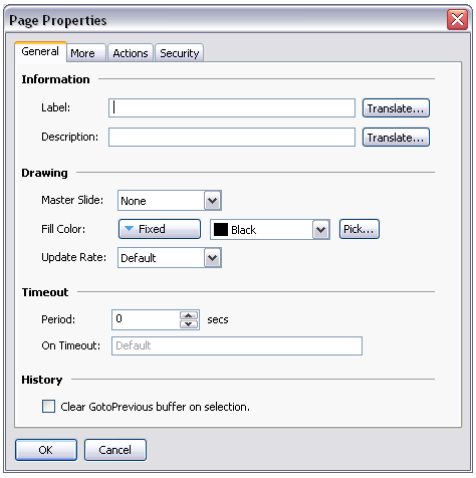
If the key is displayed in BLUE, local and global actions are BOTH defined.

Once you have defined an action, you can right-click on the key and use the resulting menu to select either Make Global or Make Local to change the action type. These options will not be available if both types of action have already been defined.

EDITING PAGE PROPERTIES

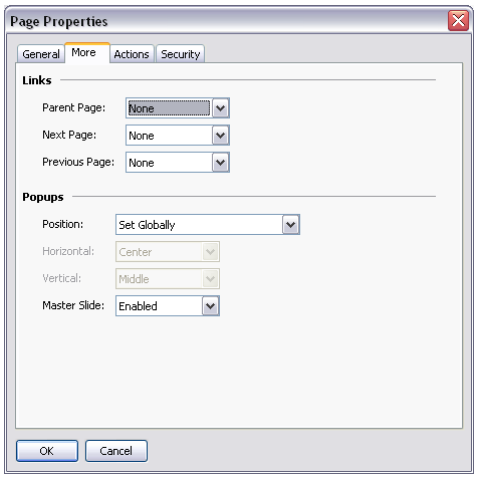
Right-clicking in the Editing Pane away from any primitives activates the context menu and allows selection of the Properties commands to edit a display page's properties…

GENERAL PROPERTIES



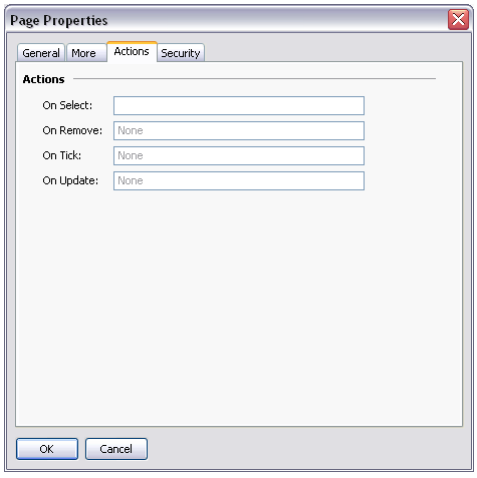
- The *Label* and *Description* properties define general-purpose translatable strings that can be accessed elsewhere using Crimson's property extraction syntax. See the chapter on Writing Expressions for more details.
- The *Master Slide* property allows the selection of another page that will be used as a background for the current page. This allows common user interface elements such as clocks, alarm status indicators and so on to be drawn on a single page and then included on several other pages.
- The *Fill Color* property defines the background color of the page, assuming that a master slide has not been used. You should avoid animating the background color, as changes will require the hardware to redraw of all items on the page, with a potential impact on performance.
- The *Update Rate* property defines the page's update rate. The overdrive setting should not be used in normal circumstances. The default setting is currently equivalent to the standard setting.
- The Timeout properties define timeout behavior. If a period of time equal to *Period* passes without user activity, the *On Timeout* action will be executed. Refer to the Writing Actions chapter for details of the possible actions.
- The *Clear GotoPrevious Buffer* property indicates that the history buffer maintained by `GotoPrevious()` and `GotoNext()` should be cleared when this page is selected. You would typically set this property on the main menu page of your database, removing the ability to go back beyond that point.

MORE PROPERTIES



- The Links property group allows a number of pages to be selected by standard actions on a display page. The *Parent Page* property defines a page to be selected if the timeout occurs and no action is defined. The *Next Page* property defines a page to be selected if input navigation is enabled and the focus is moved beyond the last field on the page. The *Previous Page* property defines a page to be selected if the focus is similarly moved beyond the first field.
- The *Position* property allows the globally defined position of popup windows to be overridden for this page. If local settings are enabled, the *Horizontal* and *Vertical* properties are used to specify the position.
- The *Master Slide* property is used to indicate whether the master slide should be kept active while a popup is displayed. The default setting of enabled allows buttons on the master slide to function, even though buttons on the actual page will be disabled while a popup is present. This can be useful if you want global navigation options on the master slide to always be available.

ACTION PROPERTIES



- The *On Select* property defines an action to be run when the page is displayed.

- The *On Remove* property defines an action to be run when the page is deselected.
- The *On Tick* property defines an action to be run once per second.
- The *On Update* property defines an action to be run on each display update.

SECURITY PROPERTIES

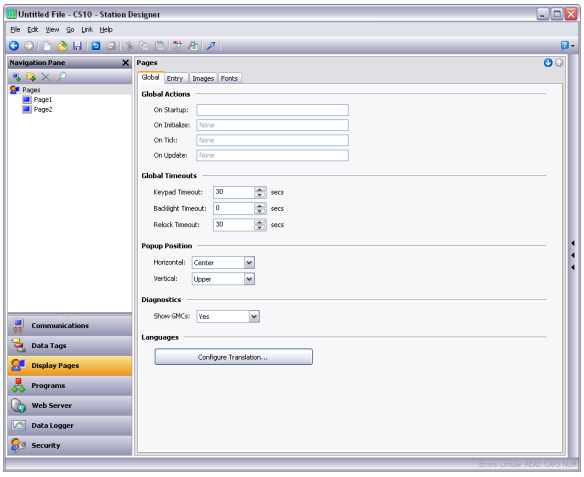
Refer to the Using Security chapter for details on security descriptors.

USER INTERFACE SETTINGS

Selecting the root item in the Navigation List will access the user interface settings.

GLOBAL PROPERTIES

The Global tab contains various general settings that apply across the database…



GLOBAL ACTIONS

- The *On Startup* property defines an action to be run when the system starts.
- The *On Initialize* property defines an action to be run slightly later[1].
- The *On Tick* property defines an action to be run once per second.
- The *On Update* property defines an action to be run on each display update.

GLOBAL TIMEOUTS

- The *Keypad Timeout* property defines the period of time without user action after which any data entry operations will be cancelled and the associated popup keypad removed from the display.

[1] The difference between these two properties is subtle, and not of concern to most users.

- The *Backlight Timeout* property defines the period of time without user action after which the display backlight will be turned off to conserve power and display life. The default value of zero disables this feature.
- The *Relock Timeout* property defines the period of time after which any actions protected via the Locked or Hard Locked methods will relock automatically, such that the user will once again have to unlock them before they can be used.

POPUP POSITION

- The *Horizontal* and *Vertical* properties define the default position for popup display pages and popup keypads. They can be overridden at the page level if desired by using the page's own properties to specify new values.

DIAGNOSTICS

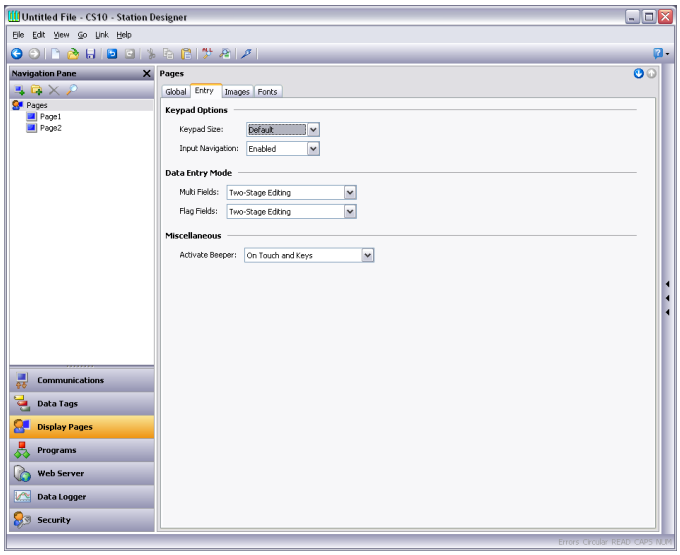
- The *Show GMCs* property is used to enable or disable the display of certain diagnostic information after a runtime system fault. The information is useful in allowing Red Lion to correct software problems, but may be distracting to users.

LANGUAGES

- The *Configure Translation* button is used to configure the languages to be used within the system. Refer to the chapter on Localization for more information.

ENTRY PROPERTIES

The Entry tab contains global settings that apply to data entry…



KEYPAD OPTIONS

- The *Keypad Size* property is used to select the size of the data entry keypad. The various settings progressively increase the size of the keypad, with a setting Maximum causing the

keypad to take up most of the screen for use in situations where, for example, operators are wearing unwieldy gloves.

- The *Input Navigation* property is used to show or hide the **NEXT** and **PREVIOUS** keys in the various popup keypads. These keys can be used to move between entry fields without first deactivating the keypad.

DATA ENTRY MODE

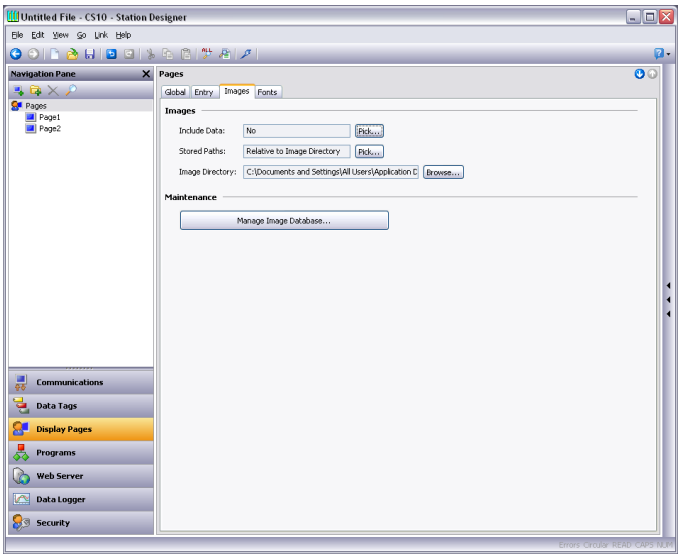
- The *Multi Data Entry* property is used to control the data entry mode used for multi-state format objects. Two-stage editing results in the **ENTER** key having to be pressed to commit any changes, while single-stage editing results in the new data being written to the associated data item as soon as **RAISE** or **LOWER** are used to make a change. Single-stage entry is faster, but may result in the writing of intermediate values when changing a multi-state setting.
- The *Flag Data Entry* property is used to control the data entry mode used for two-state format objects. It operates in the same way as the property above.

MISCELLANEOUS

- The *Activate Beeper* property is used to turn the target device's beeper on or off as desired. The beeper provides feedback as to keyboard and touch-screen activation, but can become annoying during the development process.

IMAGES PROPERTIES

The Images tab is used to manage images within the database…



IMAGES

- The *Include Data* property indicates whether external images dragged into a display page should be stored as pointers to the source location, or whether the actual image data should be included in the database file. Including image data will typically make the database very

large, and may make it impossible to use the Support Upload feature without filling the memory of the target device.

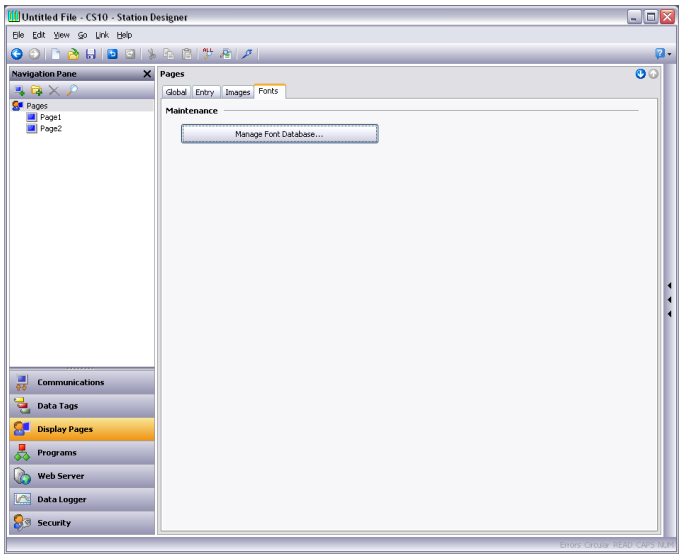
- The *Stored Paths* property defines how image links are stored. Absolute mode stores the full path, including the drive letter. The two relative modes store and interpret image paths relative to either the database or the Crimson image directory, allowing database and image files to be moved between machines without too much worry about absolute path locations.
- The *Image Directory* property defines the image path referenced above.

MAINTENANCE

- The *Manage Image Database* button is used to invoke the Image Manager in order to view and manipulate the images used in the database. See the section below for more information on this facility.

FONTS PROPERTIES

The Fonts tab is used to manage fonts within the database…



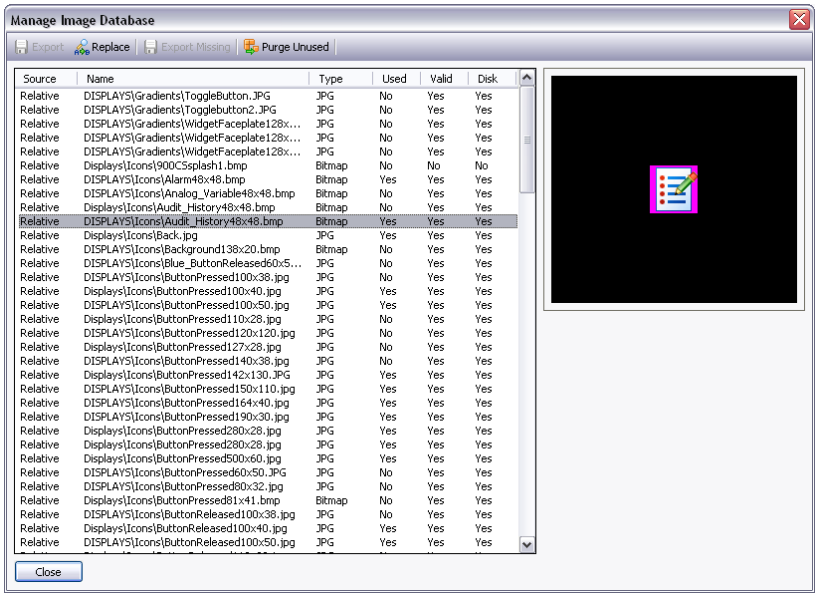
MAINTENANCE

- The *Manage Font Database* button is used to invoke the Font Manager in order to view and manipulate the fonts used in the database. See the section below for more information on this facility.

MANAGING IMAGES

The Image Manager is invoked from the Images tab of the user interface settings. It contains a list of all the images referenced in the database, together with their properties. It allows you to view the images, and to perform certain changes to how the images are stored and used.

The sample below shows the Images Manager from a complex database…



The main list view shows the properties of the various images…

- The *Source* column indicates whether the image is being obtained from a file via either a fixed or a relative path, from the Symbol Library, or from internal data stored when an image was pasted or dragged from another source.
- The *Name* column shows the filename for images stored in files, and the relevant symbol information for images sourced from the Symbol Library.
- The *Type* column shows the file type of the image data.
- The *Used* column indicates whether the image is used in the database.
- The *Valid* column indicates whether valid image data is available. This column may be set to No if an image was being sourced from a disk file that is no longer available, and if the database is not configured to hold its own image data via the Include Data property described above.
- The *Disk* column indicates whether the image exists on disk. Images that were pasted or dragged directly into the editor may not ever have existed on disk, and images sourced from files but also stored within the database itself may now be missing if the file is no longer available.

The toolbar at the top of the window allows various commands to be performed…

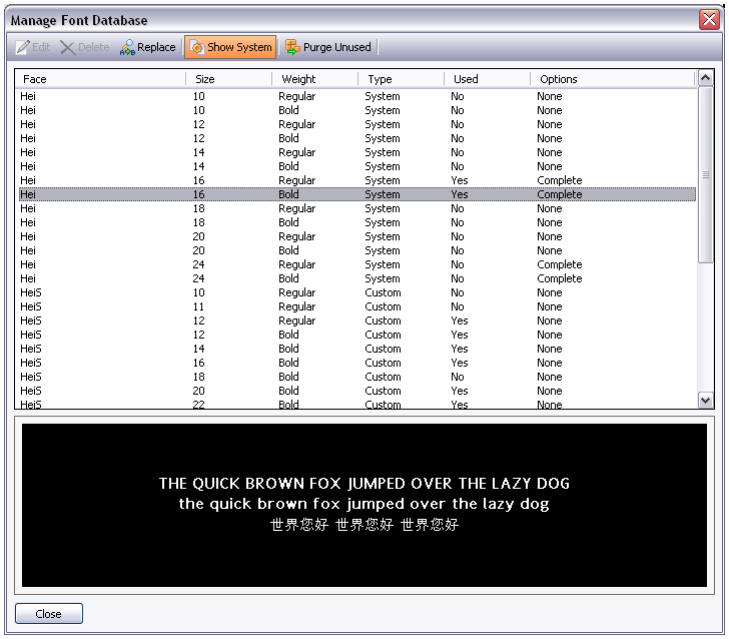
- The *Export* command saves an image that is available but not stored on disk to a file. If a filename has already been defined for the selected image, that name will be used. In other cases, you will be prompted to select a filename.
- The *Replace* command allows you to replace a given image with another. All references to the image in the database will be updated to reflect the change.

- The *Export All* command saves all images that are available but not stored on disk and that have filenames defined. It can be used to ensure that all images are stored in external files prior to turning off Include Data.
- The *Purge Unused* command is used to remove all images that are not used in the database, thereby saving disk space when saving the database to disk. Use of this command may also reduce memory usage in the target device.

MANAGING FONTS

The Font Manager is invoked from the Fonts tab of the user interface settings. It contains a list of all the fonts referenced in the database, together with their properties. It allows you to view the fonts, and to perform certain changes to how the fonts are stored and used.

The sample below shows the Font Manager from a complex database…



The main list view shows the properties of the various fonts…

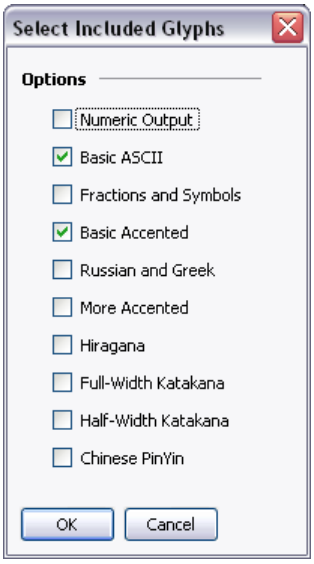
- The *Face* property shows the name of the font.
- The *Size* property shows the height in pixels of the font.
- The *Weight* property indicates whether the font is bold or not.
- The *Type* property indicates whether the font is a system or custom font.
- The *Used* property indicates whether the font is used in the database.
- The *Options* property lists the options selected for the font.

The toolbar at the top of the window allows various commands to be performed…

- The *Edit* button allows the properties of custom fonts to be edited.

- The *Delete* button allows an unused font to be deleted. Once a font is deleted, it will no longer be presented in the drop-down used for font selection, but may be recreated by using the associated Pick button.
- The *Replace* button allows a font to be replaced with another. All references to the font in the database will be updated to reflect the change.
- The *Show System* button controls whether system fonts are shown in the list.
- The *Purge Unused* button removes all unused fonts from the database, thereby reducing the amount of memory used in the target device. As with a deleted font, purged fonts will no longer be presented in the drop-down list used for font selection, but may be recreated by using the associated Pick button.

Editing the properties of a custom font produces the following dialog box…



The various options allow specific sets of characters to be included in the font image that is created and downloaded to the target device. Restricting the characters to the ones that are needed for your application will save memory, especially with larger fonts. Note that the Numeric Output option can be used alone to restrict the font to digits, decimal points and those other characters used to render conventional, scientific or hexadecimal numbers.

PRIMITIVE TYPES

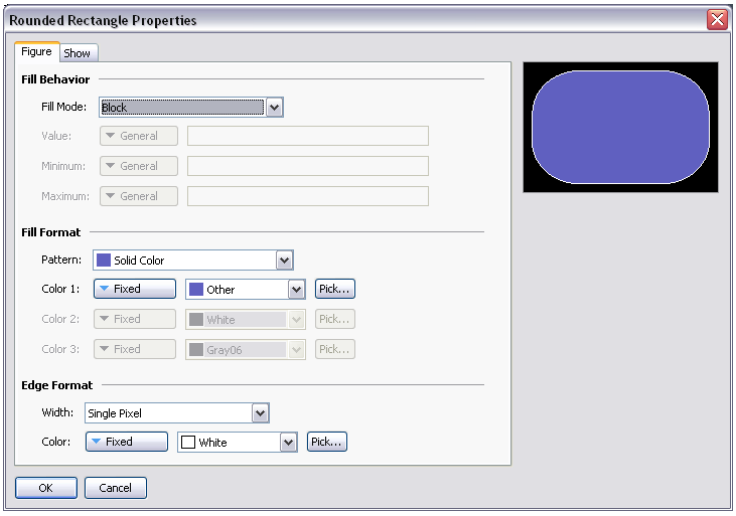
This chapter describes each of the primitives provided by Crimson.

CORE PRIMITIVES

GEOMETRIC PRIMITIVES

The geometric primitives represent simple shapes: a rectangle, a circle, a trimmed rectangle, a rectangle with rounded corners, a plaque and a wedge. All these primitives support tank fills, and can therefore be used to implement effects such as bar graphs. They also support the addition of text or data, and can therefore be used to create text or data displays, or to provide data entry. Finally, they support the addition of actions, and can therefore be used to implement interactive display elements.

The primitive-specific property tab for these primitives is shown below…



Refer to the previous chapter for details of the standard fill and edge settings. Note that the wedge has one additional property, namely a *Position* property used to specify the orientation of the triangular wedge within the bounding rectangle.

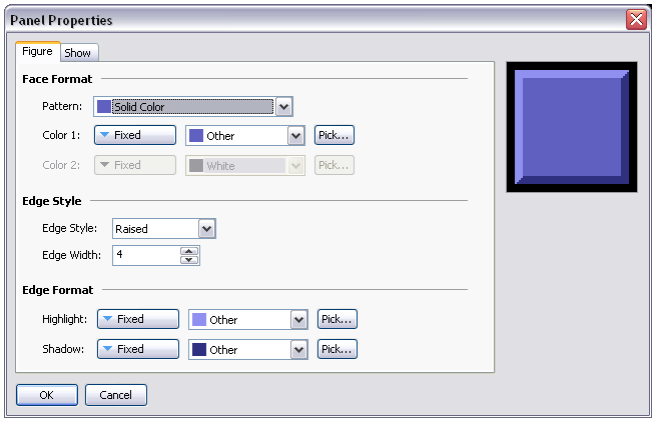
The trimmed rectangle, rectangle with rounded corners and plaque all have a layout handle that can be used to specify the radius of the corner effect. In their degenerate form with a zero corner radius, they become equivalent to a simple rectangle.

While the geometric primitives are very simple, their support for tank fills, data, text and actions means that a large portion of most databases can in fact be created by using just the rectangle or the rounded rectangle.

3D PRIMITIVES

The various 3D primitives represent rectangles with a three-dimensional border. While three versions are presented in the Resource Pane, all are really just preconfigured variations of a single primitive. These primitives support tank fills, and can therefore be used to implement effects such as bar graphs. It also supports the addition of text or data, and can therefore be used to create text or data displays, or to provide data entry. Finally, it supports the addition of actions, and can therefore be used to implement interactive display elements.

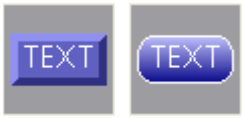
The primitive-specific property tab for these primitives is shown below…



Refer to the previous chapter for details of the standard fill and edge settings. The *Edge Style* selects the type of edge to be drawn, effectively choosing between the three predefined versions shown above. The *Edge Width* property defines the number of pixels to be allocated to each edge element. Primitives with an edge style of Border will have an edge that is sized to twice the defined edge width.

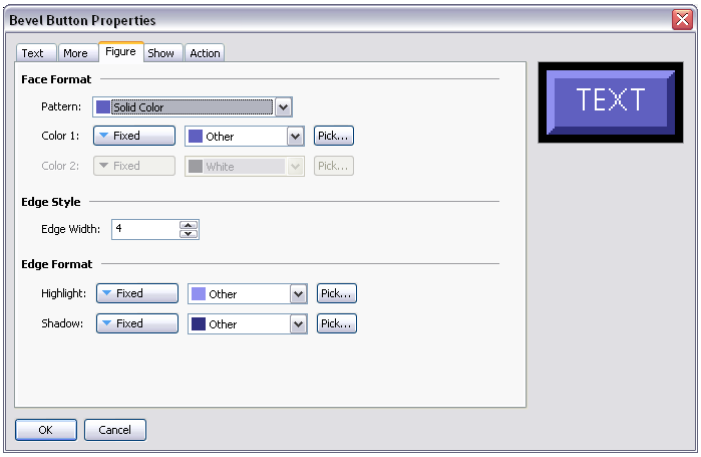
As with the geometric primitives, the 3D primitives can be used to create a large portion of a standard database by virtue of their support for tank fills, data, text and actions.

BUTTON PRIMITIVES



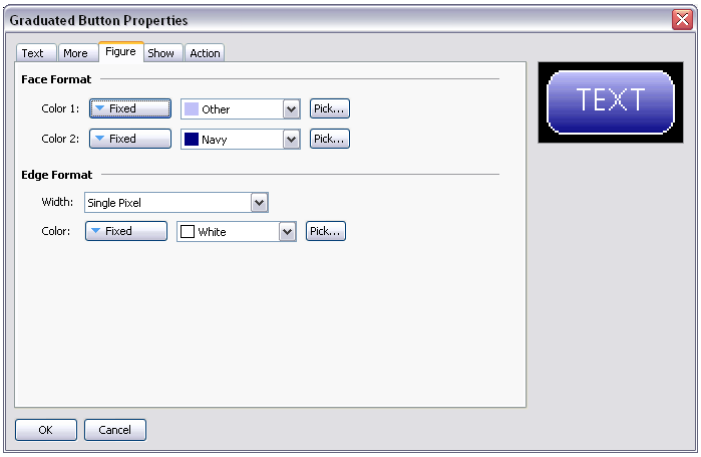
The button primitives implement beveled or graduated buttons. Text is preconfigured to allow the button to be labeled, but can be removed to allow the addition of live data. An action tab is also provided by default, but will be disabled if live data is added and configured for data entry. Buttons with data entry fields use the button press to activate editing.

The primitive-specific property tab for a beveled button is shown below…



Refer to the previous chapter for details of the standard settings. The *Edge Width* property defines the number of pixels to be allocated to each edge element. Since this primitive always uses an edge style of Border, the edge will be sized to twice the defined edge width.

The primitive-specific property tab for a graduated button is shown below…



Refer to the previous chapter for details of the standard settings.

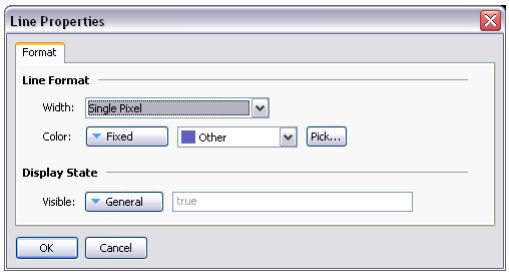
TEXT AND DATA PRIMITIVES

The text box and data box primitives are in fact rectangles with predefined data and text items, and with no fill or edge colors defined. They exist to make it easier to add data and text elements, and to provide comfort to those users who are not used to being able to construct an entire database from simple geometric primitives! They can also be used to add a second data or text element to a primitive or when constructing a group.

Refer to the previous sections for details of the standard settings.

LINE PRIMITIVE

The line primitive implements a simple line. The property dialog is shown below…

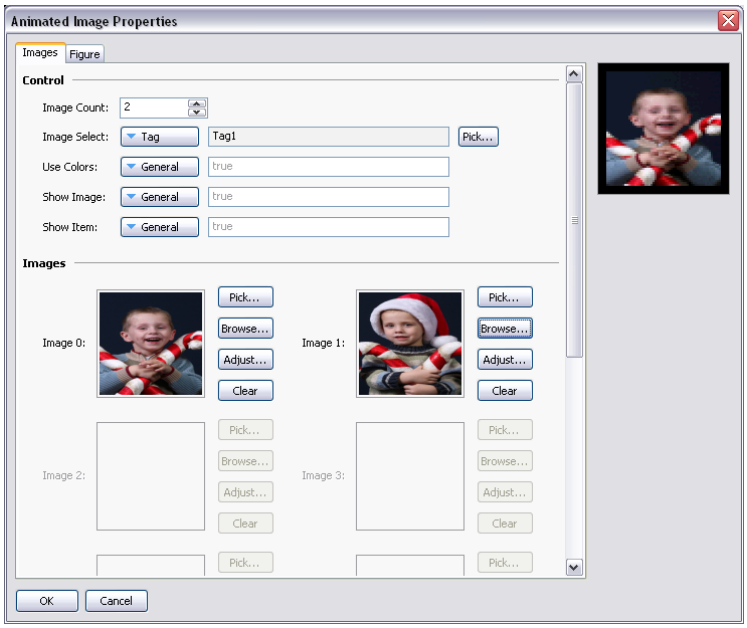


Refer to the previous chapter for details of the standard settings.

IMAGE PRIMITIVE

The image primitive is used to display an image, possibly chosen from a number of images based upon a numeric value. The primitive supports the display of bitmaps, JPEGs, metafiles, bitmaps and many other image types. It can operate with a transparent or filled background, and can optionally define an edge to go around the image. It also supports the addition of data, text or actions, thereby allowing more complex elements to be constructed.

The Image tab for an animated image primitive is shown below…



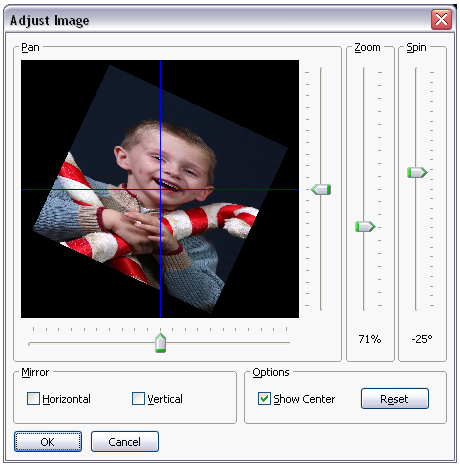
- The *Image Count* property defines the number of image slots that are defined for this primitive. One of the images will be chosen for display at any given time, based upon the value of the Image Select property.
- The *Image Select* property selects the desired image. It is treated as a zero-based value and is reduced modulo the Image Count. In other words, if four images are defined, values of 0, 4, 8 etc. will display the first image, values of 1, 5, 9 etc. will display the second image and so on.
- The *Use Color* property is used to either reduce an image to black-and-white or to preserve its color. An expression that evaluates to a non-zero value or an empty expression will result in a color image. A zero value will reduce the image to grayscale using standard r-g-b brightness weightings. This option is useful when showing the disabled state of an image on a button.
- The *Show Image* property is used to show or hide the image. If the primitive has no edge or fill defined, it is functionally equivalent to the *Show Item* property, but will otherwise still display the edge or background as per the configuration.
- The *Show Item* property is used to show or hide the entire primitive.

DEFINING IMAGES

The *Images* section of the dialog box defines the images for each slot. The Pick button next to each image will display a dialog box reminding you that you can simply drag an image onto the field. This image can be dragged from the Symbol Library category in the Resource Pane, from a folder in Windows Explorer or from any other drag-and-drop capable application. The Browse button can be used to open a file containing a suitable image format and to load that file into this image slot. As mentioned above, JPEGs, metafiles, bitmaps and many other file formats are supported.

ADJUSTING IMAGES

The Adjust button next to the image can be used to modify the image…



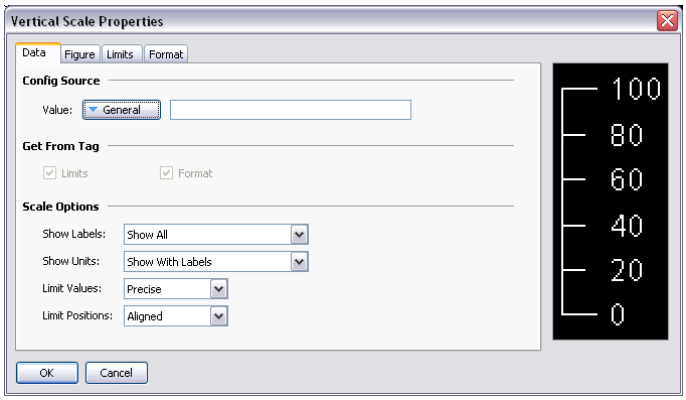
The various sliders can be used to pan, zoom and spin the image, while the checkboxes can be used to mirror it horizontally or vertically. The Show Center checkbox shows or hides the blue lines that mark the center of the image, while the Reset button can be used to restore the

image to its original state. The manipulation options are sometimes used to modify an image so as to create various different states for use in animation.

SCALE PRIMITIVE

The scale primitive is used to draw a vertical scale. The limits of the scale can be defined as either constants, or can be varied according to the value of specific expressions. The scale can be labeled or unlabelled, with any labels being based upon a specified format object which may optionally be obtained from a tag.

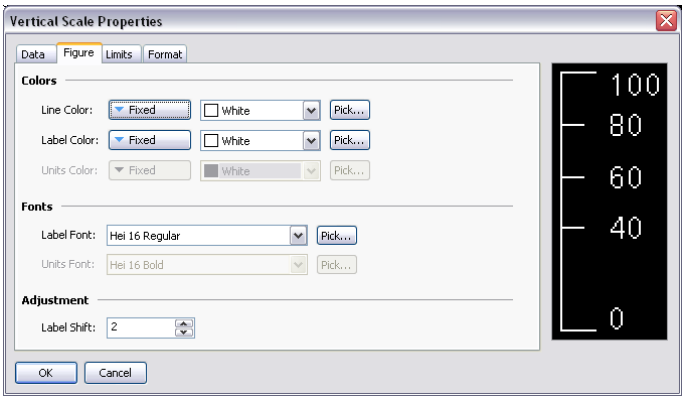
DATA PROPERTIES



- The *Value* property defines an optional tag that will be used to obtain limits and format information for the scale. The value itself is not actually used by the primitive, but the tag is simply used as a source for further information.
- The *Get From Tag* properties are used to indicate whether the tag optionally defined in the Value property should be used as a source for the data in question.
- The *Show Labels* property is used to show or hide the numeric scale labels.
- The *Show Units* property is used to show or hide the units defined by a numeric data format. The units may be appended to each scale label, or may be drawn vertically by the edge of the scale.
- The *Limit* Values property specifies how the top and bottom values of the scale are determined. If a setting of Precise is specified, the limit values will be used exactly, even if this produces limits that do not exactly correspond to the automatically selected tick spacing. This can produce irregular scale labels, but will ensure that a tank fill placed next to the scale and bound to the same tag will be drawn exactly as required by the scale primitive. A setting of Rounded allows the scale primitive to automatically adjust the limits to achieve regularly spaced tick marks.
- The *Limit Positions* property specifies how the limits of the scale relate to the unit labels. A setting of Aligned keeps the tick marks and the labels aligned precisely, at the cost of moving

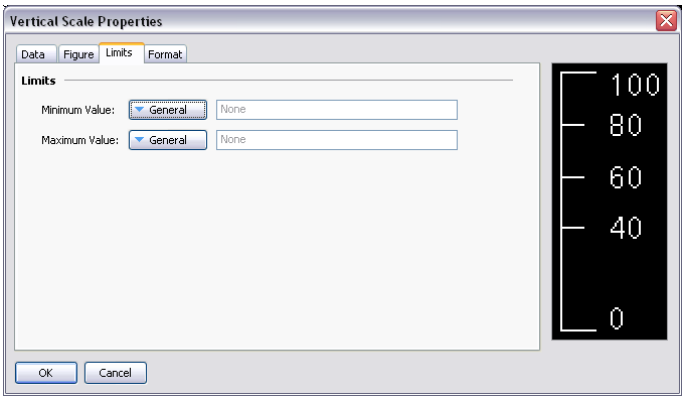
the outer tick marks inwards from the edge of the primitive. Choosing a setting of Shifted moves the outer two labels relative to the tick marks, but allows the minimum and maximum ticks to line up with the edge of the primitive, making it easier to align with, say, a tank fill.

FIGURE PROPERTIES



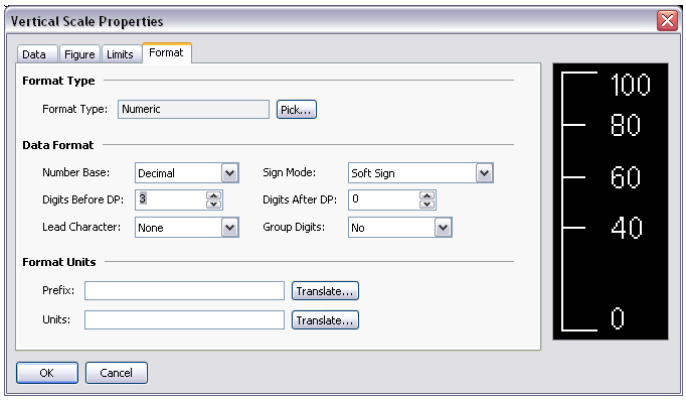
The properties on this page define the colors and fonts used for the scale. Refer to the previous chapter for details of the standard properties. The *Label Shift* property can be used to move the labels up or down relative to the tick marks, producing more attractive results when working with fonts with spacing above or below the character glyphs.

LIMIT PROPERTIES



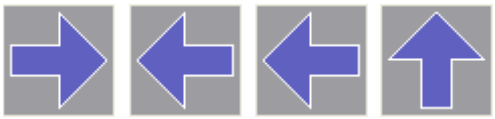
The properties on this page are used to set the minimum and maximum values to be shown by the scale. Expressions may be specified, in which case Crimson will dynamically update the scale at runtime, choosing tick marks and label positions appropriate to the new values. These settings may not be available if a tag has been chosen as the source of the limit values.

FORMAT PROPERTIES



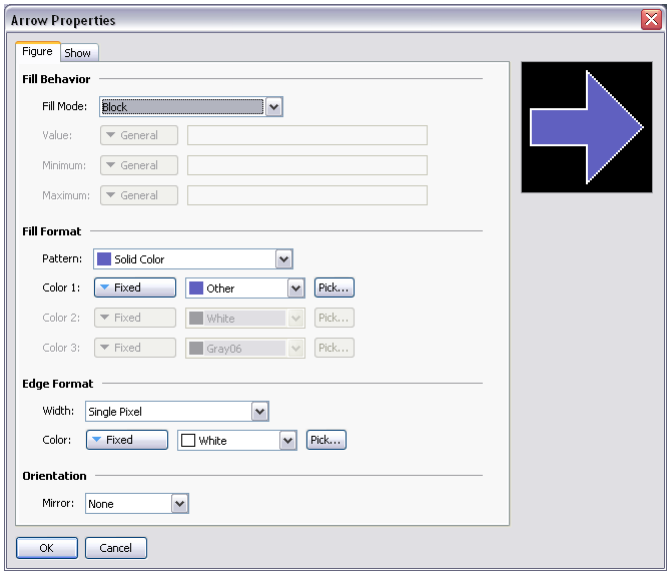
The *Format Type* field specifies the format type to be used when drawing the scale labels. Only general or numeric formats are supported. The selection may not be available if the format is being obtained from a tag. Refer to the section on Using Formats for details of the various other properties that are displayed when a numeric data format is selected.

ARROWS



The four arrow primitives are in fact predefined versions of a single primitive. This primitive supports tank fills. It also supports the addition of text or data, and can therefore be used to create text or data displays, or to provide data entry. Finally, it supports the addition of actions, and can therefore be used to implement interactive display elements.

The primitive-specific property tab is shown below…



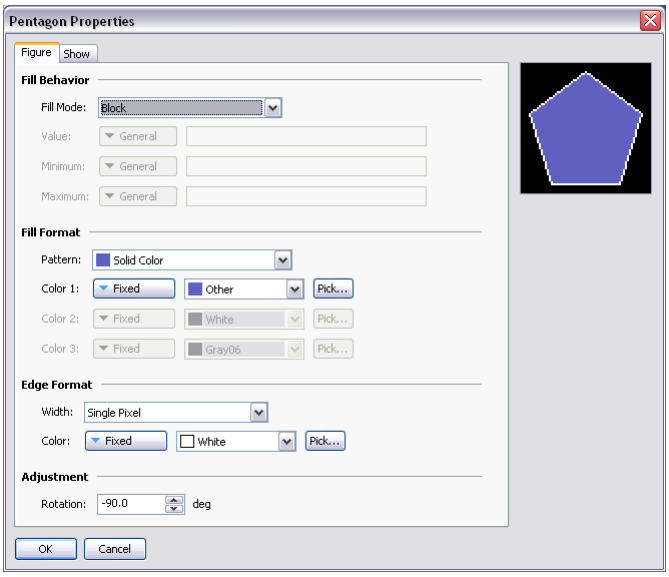
Refer to the previous chapter for details of the standard settings. The *Mirror* property is used to control the direction of the arrow. It is this property that is used to produce the four predefined versions shown in the Resource Pane.

POLYGONS AND STARS

POLYGONS

These primitives as used to display regular polygons: a triangle, a pentagon, a hexagon and an octagon. All support tank fills. They also support the addition of text or data, and can therefore be used to create text or data displays, or to provide data entry. Finally, they support the addition of actions, and can therefore be used to implement interactive display elements.

The primitive-specific property tab for these primitives is shown below…

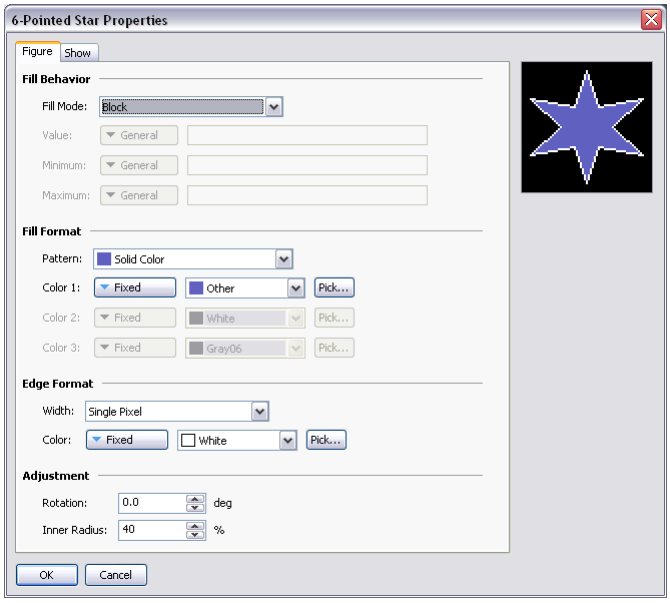


Refer to the previous chapter for details of the standard fill and edge settings. The *Rotation* property can be used to rotate the polygon within the bounding rectangle. The x- and y-axes are scaled such that the overall width and height of the polygon fill the rectangle.

STARS

These primitives represent regular stars with four, five, six and eight points. All these primitives support tank fills. They also support the addition of text or data, and can therefore be used to create text or data displays, or to provide data entry. Finally, they support the addition of actions, and can therefore be used to implement interactive display elements.

The primitive-specific property tab for these primitives is shown below…



Refer to the previous chapter for details of the standard fill and edge settings. The *Rotation* property can be used to rotate the polygon within the bounding rectangle. The x- and y-axes are scaled such that the overall width and height of the polygon fill the rectangle. The *Inner Radius* property is used to change the pointedness of the star. (Stars are created by taking a regular polygon with *2n* sides, and by then changing the radius between alternate points as the polygon is drawn. This property controls the ratio of the radii.)

BALLOONS AND CALL-OUTS

The balloon primitive provided can be used to label items on a page or to provide help to operators. It supports the addition of both text and data, and, for what it is worth, can also be configured to show a tank fill. It also supports the addition of actions, and can therefore be used to implement interactive display elements.

The exact design of the balloon is controlled via a number of layout handles…

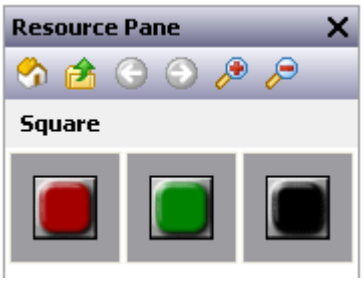
The top-left handle controls the radius of the corners. The center handle controls the height of the balloon body relative to the balloon tail. The bottom handle controls the position of the balloon tail. Text within the balloon will be automatically reflowed as the handles are moved.

SEMI-TRIMMED FIGURES

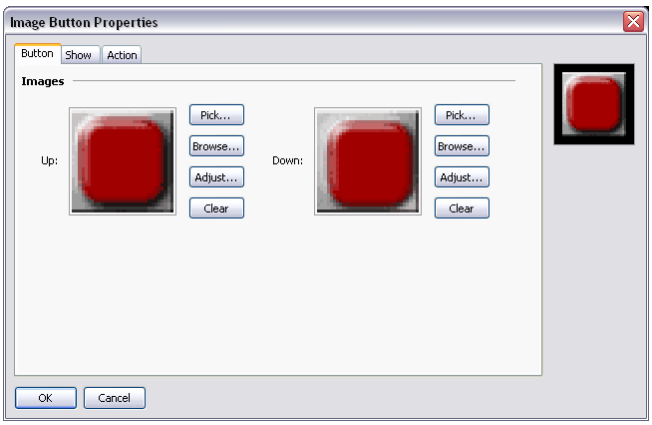
The semi-trimmed figures are versions of the rounded rectangle, trimmed rectangle and plaque that have only two of their corners removed. These are useful for creating title bars and other effects on the edge of primitive groups. They are each available in four orientations.

ACTIONS BUTTONS

Actions buttons use preselected images from the Symbol Library to create a button that will perform a given action when it is pushed. Many versions are provided beyond those shown above. Clicking on a given button in the Resource Pane will show the different color variants that are available. For example, the square button is available in red, green or black…



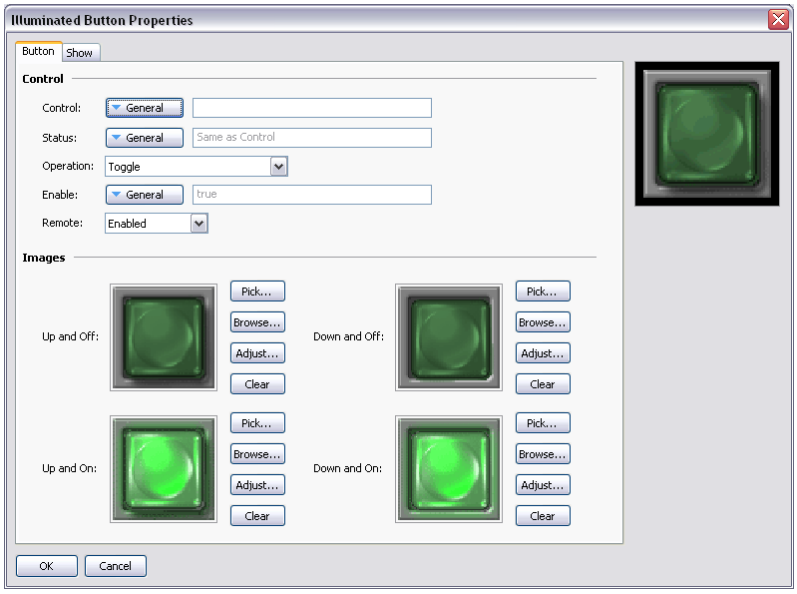
When using an action button, you will primarily use the Action tab of the properties dialog to define an action as per the description in the previous chapter. The Button tab can also be used to adjust the button images, or to define your own versions…



ILLUMINATED BUTTONS

Illuminated buttons use preselected images from the Symbol Library to create a button that will control a tag, and light up based either upon the state of that tag or the state of another expression. Many versions are provided beyond those shown above. Clicking on a given button in the Resource Pane will show the different color variants that are available. For example, the candy button shown above is available in red, green, yellow, blue or grey…

The primitive-specific property tab for these primitives is shown below…



- The *Control* property defines the value to be written when the button is pressed or released. This value must be writable, and will be set to one or zero depending on the exact operation defined for the button.
- The *Status* property is used to control the illumination of the button. If it is left blank, it will default to the Control value. A different value can be used if more complex logic is required.

- The *Operation* property selects the required behavior…

OPERATION	BUTTON BEHAVIOR
Toggle	Change the data state when the button is pressed.
Latching	If the data is 0, set it to 1 when the button is pressed. If the data is 1, set it to 0 when the button is released.
NO Momentary	Set the data to 1 when the button is pressed. Set the data to 0 when the button is released.
NC Momentary	Set the data to 0 when the button is pressed. Set the data to 1 when the button is released.
Turn On	Set the data to 1 when the button is pressed.
Turn Off	Set the data to 0 when the button is pressed.
Custom	The behavior is defined using the Action tab.

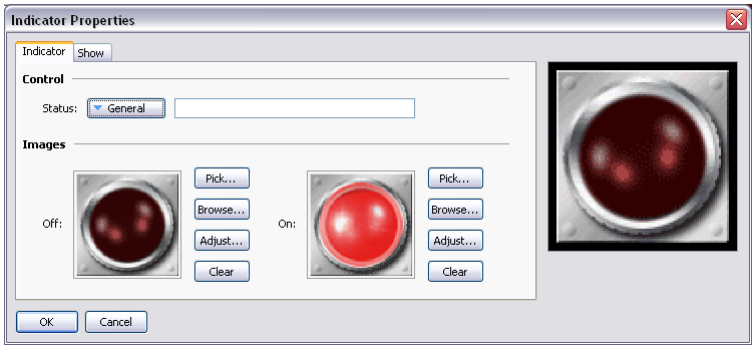
Note that Latching is slightly different from Toggle in respect of the point at which a non-zero control value is set back to zero. Toggle makes all changes when the button is pressed, while Latching turns a value off when it is released. This produces a result more in keeping with the behavior of a real-world latching pushbutton.

Refer to the previous chapter for details of the *Protection*, *Enable* and *Remote* properties, and to earlier in this chapter for details of how to change or adjust the various button images. As you can see from the example above, four images are required to represent the button states.

INDICATORS

Indicators use preselected images from the Symbol Library to show the on/off status of a data value. Many versions are provided beyond those shown above. Clicking on a given button in the Resource Pane will show the different color variants that are available. For example, the pilot indicator shown above is available in red, green, yellow, blue or white…

Indicators have a very simple set of properties…

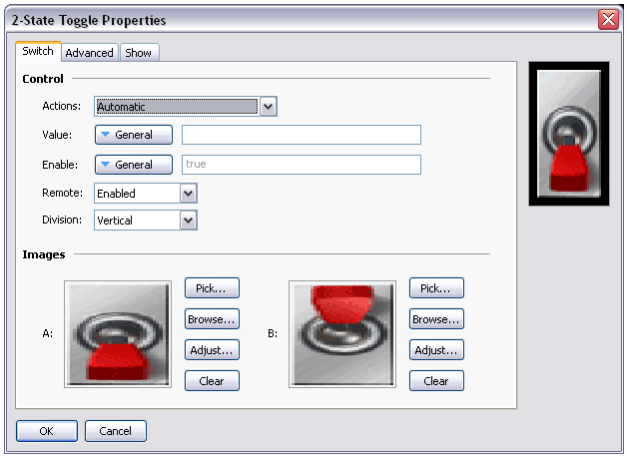


The *Status* property controls the images to be drawn. All other properties are standard.

2-STATE TOGGLES

2-State Toggles use preselected images from the Symbol Library to implement toggle switches with up and down positions. Many versions are provided beyond those shown above. Clicking on a given toggle in the Resource Pane will show the different color variants that are available. For example, the paddle switch is available in red, green or black…

SWITCH PROPERTIES

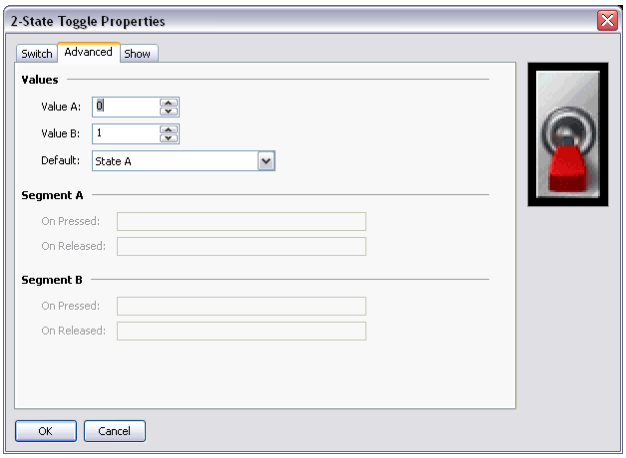


- The *Actions* property controls the behavior of the switch. The three automatic modes model conventional or biased toggle switches, while User-Defined mode allows you to specify more complex actions that will occur when either half of the toggle switch is pressed or released.
- The *Value* property is used in the automatic modes and will be written to the data values associated with States A and B as the switch is changed. By default, State A is represented by a zero and State B by a one, but these values can be changed using the advanced settings for this primitive.
- The *Divisions* property defines whether the switch is thrown vertically or horizontally, and therefore how Crimson should divide the primitive when interpreting touches by the user.

Refer to the previous chapter for details of the *Protection*, *Enable* and *Remote* properties.

Refer to earlier in this chapter for details of how change or adjust the switch images.

ADVANCED PROPERTIES

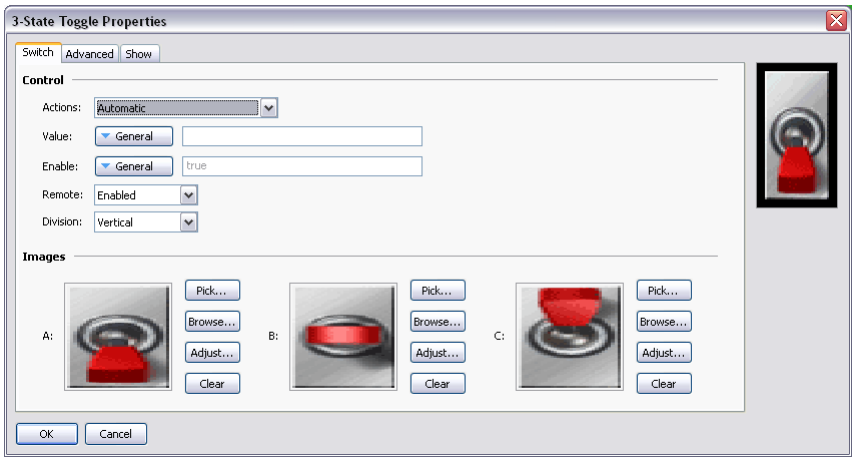


- The *Value A* and *Value B* properties define the data values used in the automatic modes to represent the two states of the switch. The value read from the Value property will be compared to these two values to decide which state to display, and changing the switch will similarly write the appropriate value.
- The *Default* property selects the state to be displayed if the data read from the Value property does not match either Value A or Value B.
- The *On Pressed* and *On Released* properties define custom behaviors to be carried out when the A and B portions of the switch are pressed or released by the user. For a vertical switch, A is the bottom half and B is the top half. For a horizontal switch, A is the left half and B is the right half.

3-STATE TOGGLES

3-State Toggles use preselected images from the Symbol Library to implement toggle switches with up, center and down positions. Other versions are provided beyond those shown above. Clicking on a given toggle in the Resource Pane will show the different color variants that are available. For example, the paddle switch is available in three colors…

SWITCH PROPERTIES

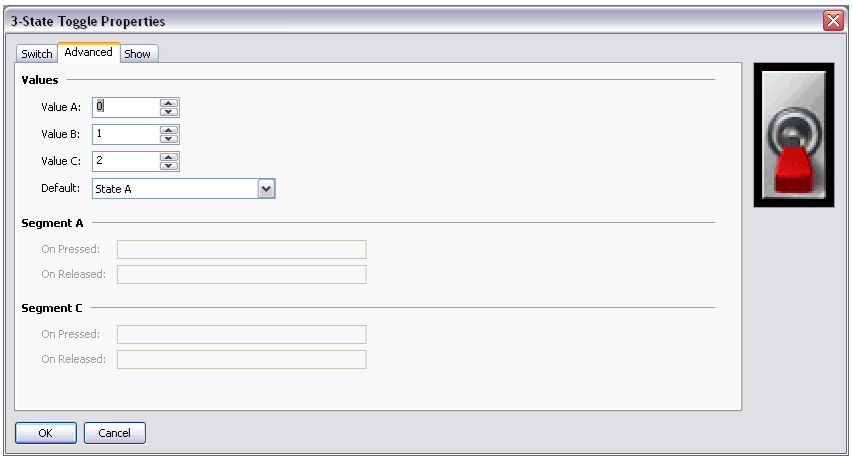


- The *Actions* property controls the behavior of the switch. The four automatic modes model conventional or biased toggle switches, while User-Defined mode allows you to specify more complex actions that will occur when either half of the toggle switch is pressed or released. Note that the switch can only be moved one position at a time, so moving from State A to State C will necessarily mean moving through State B, as would occur with a physical toggle switch.
- The *Value* property is used in the automatic modes and will be written to the data values associated with States A, B or C as the switch is changed. By default, State A is represented by a zero, State B by a one and State C by a two, but these values can changed using the advanced settings for this primitive.
- The *Divisions* property is used whether the switch is thrown vertically or horizontally, and therefore how Crimson should divide the primitive when interpreting touches by the user.

Refer to the previous chapter for details of the *Protection*, *Enable*, *Remote* properties.

Refer to earlier in this chapter for details on how to change or adjust the switch images.

ADVANCED PROPERTIES



- The *Value A*, *Value B* and *Value C* properties define the data values used in the automatic modes to represent the three states of the switch. The value read from the Value property will be compared to these values to decide which state to display, and changing the switch will write the appropriate value.
- The *Default* property selects the state to be displayed if the data read from the Value property does not match Value A, Value B or Value C.
- The *On Pressed* and *On Released* properties define custom behaviors to be carried out when the A and C portions of the switch are pressed or released by the user. For a vertical switch, A is the bottom half and C is the top half. For a horizontal switch, A is the left half and C is the right half.

2-STATE SELECTORS

2-State Selectors use preselected images from the Symbol Library to implement rotary selector switches with two states. Their behavior is identical to Two-State Toggles, and they are in fact implemented using the same primitive.

3-STATE SELECTORS

3-State Selectors use preselected images from the Symbol Library to implement rotary selector switches with three states. Their behavior is identical to Three-State Toggles, and they are in fact implemented using the same primitive.

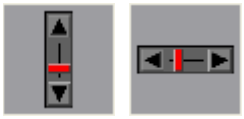
LEGACY PRIMITIVES

These primitives are provided for compatibility with other software packages.

ELLIPSE FRAGMENTS

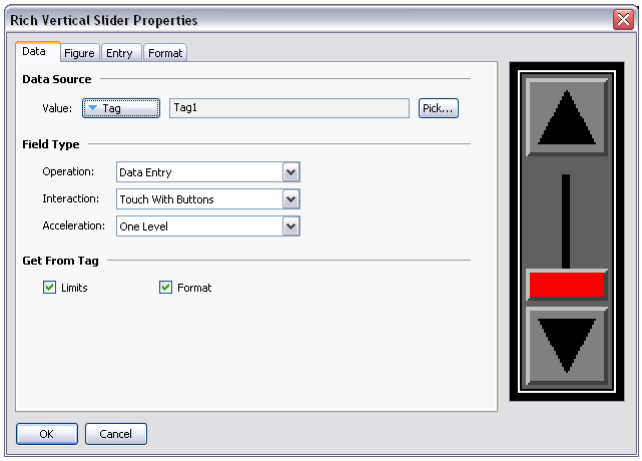
These primitives represent quarter or half of an ellipse. Their properties are conventional.

RICH SLIDERS



Rich Slider primitives allow a tag value to be adjusted by means of an analog slider. While they can be useful, they are likely to be superseded by more powerful primitives in a later release of Crimson and are thus in the Legacy sub-category.

DATA PROPERTIES



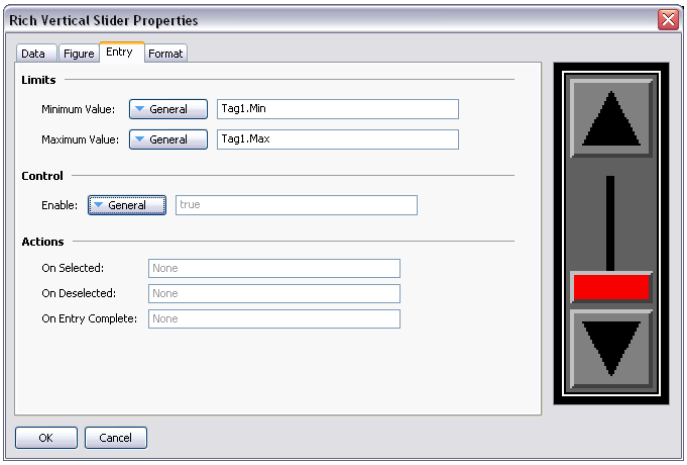
- The *Value* property specifies the data whose value is to be edited.
- The *Operation* property is used to indicate whether data entry is to be enabled or not. The default value enables entry, as read-only sliders tend to mislead.
- The *Interaction* property specifies how the user will interact with the primitive, be it via the push buttons, by manipulating the slider directly, or by both methods.
- The *Acceleration* property specifies how many levels of acceleration will be provided during data entry. Acceleration moves the slider progressively quicker after an appropriate number of steps have been taken. More than one level of acceleration can result in large changes being made inadvertently.

- The *Get From Tag* properties are used to indicate whether the slider limits and data format will be obtained from the tag provided in the Value property or whether they will be entered manually.

FIGURE PROPERTIES

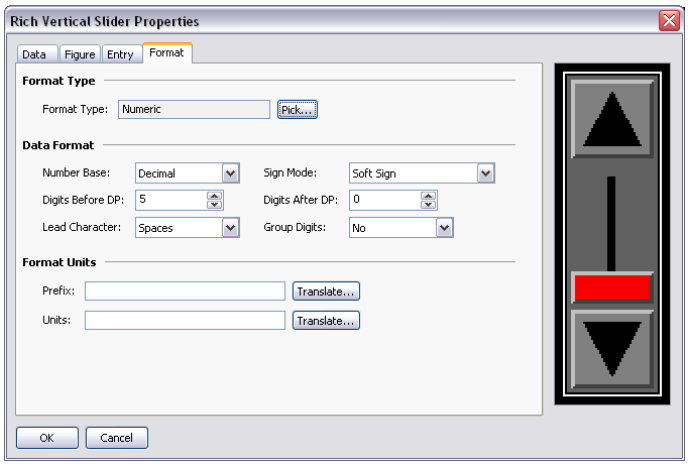
Refer to the previous chapter for details of the standard fill and edge settings.

ENTRY PROPERTIES



Refer to the previous chapter for details of the standard data entry properties.

FORMAT PROPERTIES

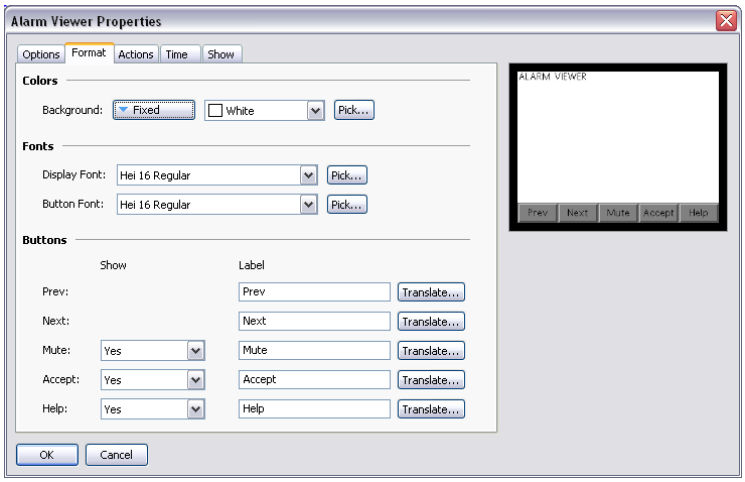


The Format tab defines the data format used by the primitive. Since the primitive doesn't actually display any data, you may wonder why it is needed, and the answer is acceleration: The acceleration of data entry depends on knowing the number base of the data being edited and the position of any decimal point. The other settings are ignored. Note that the format selection may not be available if the format is being obtained from the controlling tag.

SYSTEM PRIMITIVES

VIEWER FORMAT

Most system primitives display or manipulate data created or accessed by Crimson. Each viewer consists of a viewing area with a number of buttons beneath. The appearance of the list-based viewers is controlled via the Format tab of the properties dialog…

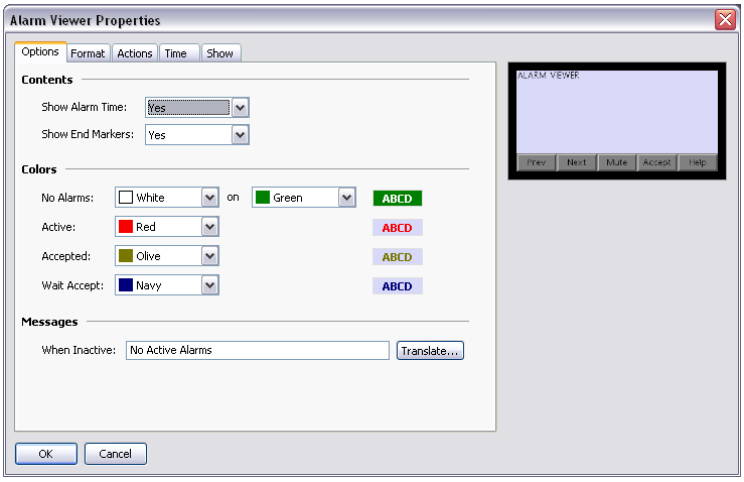


Colors and fonts are specified in the conventional way. The Buttons allows some of buttons at the bottom of the viewer to be disabled, or to allow their labels to be edited or translated for international applications. Remember that translatable strings can be set to expressions, implying that the label on a button can be customized at runtime.

ALARM VIEWER

The Alarm Viewer is used to display and optionally accept alarms within the system.

OPTION PROPERTIES



- The *Show Alarm Time* property is used to indicate whether each alarm should be prefixed with the time and date at which it occurred. The exact time format to be used is specified on the Time tab.
- The *Show End Markers* property is used to indicate whether markers should be included in the list to flag the first and last items, thereby making it easier for the user to know when they are at either end of the list.
- The *Colors* property group specifies the text colors to be used when showing alarms in different states. The No Alarms message allows a dedicated background color to be defined, while the various state-specific colors always use the background of the primitive itself.
- The *When Inactive* property defines or perhaps translate the string that is displayed by the primitive when no alarms are active.

ACTIONS PROPERTIES

If the Help button at the bottom of the viewer is enabled via the Format tab, the *On Help* action defines an action to be executed when the button is pressed.

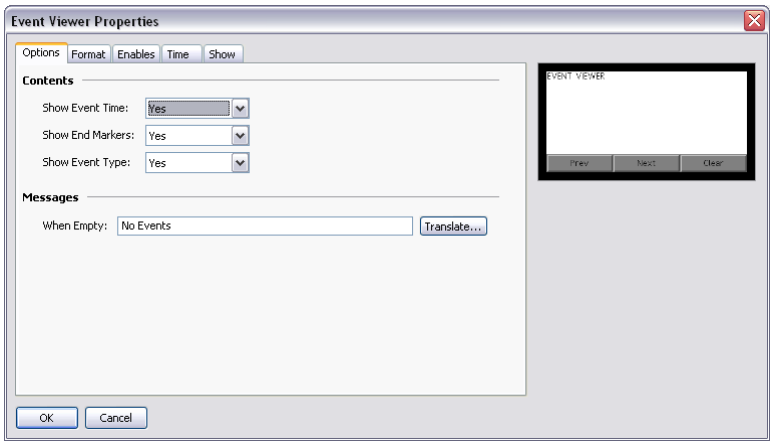
TIME PROPERTIES

The Time tab defines the format to be used when indicating the time and date at which an alarm occurred. Refer to the chapter on Using Formats for detailed information.

EVENT VIEWER

The Event Viewer is used to view and optionally clear the events logged by the system in response to alarms or events generated by data tags.

OPTIONS PROPERTIES



- The *Show Event Time* property is used to indicate whether each event should be prefixed with the time and date at which it occurred. The exact time format to be used is specified on the Time tab.
- The *Show End Markers* property is used to indicate whether markers should be included in the list to flag the first and last items, thereby making it easier for the user to know when they are at either end of the list.
- The *Show Event Type* property is used to indicate whether each entry should be marked to indicate whether it is an alarm occurrence, acceptance or clearance, or whether it represents a simple event. If alarms are in use, failing to enable this setting can produce rather confusing displays.
- The *When Empty* property defines or perhaps translate the string that is displayed by the primitive when no events are present in the log.

ENABLES PROPERTIES

If the Clear button at the bottom of the viewer is enabled via the Format tab, the *Enable Clear* property is used to enable or disable the clearing of the event log.

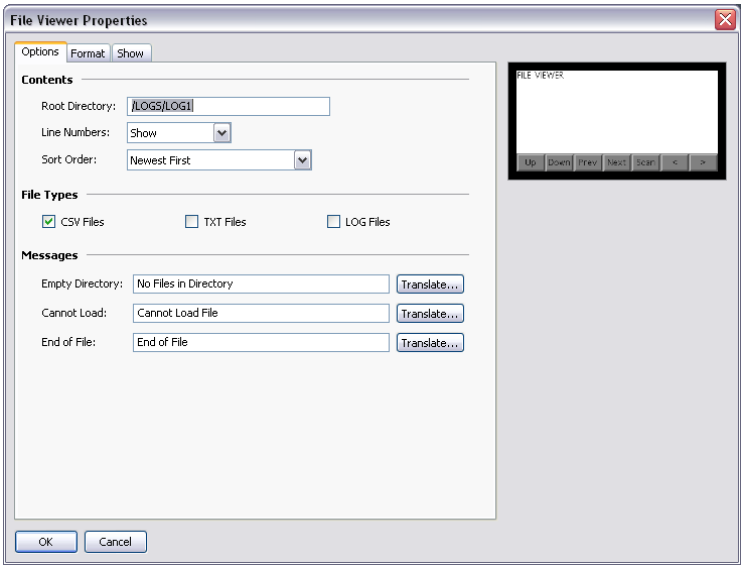
TIME PROPERTIES

The Time tab specifies the format to be used when indicating the time and date at which an event occurred. Refer to the chapter on Using Formats for more details.

FILE VIEWER

The File Viewer is used to allow the user to view text files on the CompactFlash card.

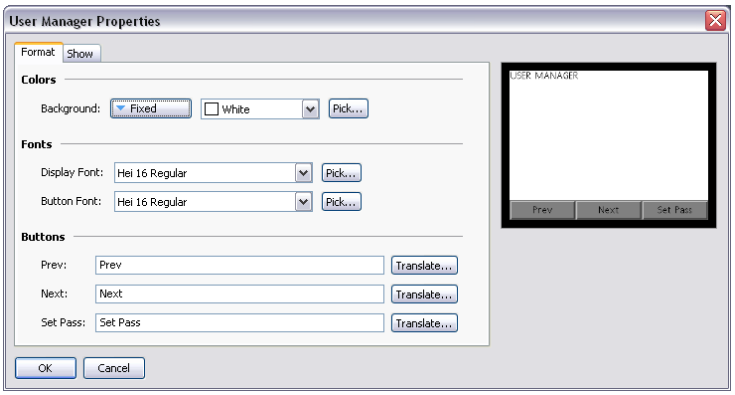
OPTIONS PROPERTIES



- The *Root Directory* property specifies the directory to be displayed.
- The *Line Numbers* property is used to show or hide line numbers on the file.
- The *Sort Order* property is used to indicate how files should be accessed.
- The *File Types* property group is used indicate the types of file that should be made available for viewing. Note that only text files can be displayed.
- The *Messages* property group defines and perhaps translate various messages used by the file viewer.

USER MANAGER

The User Manager is used to allow the changing of passwords at runtime…

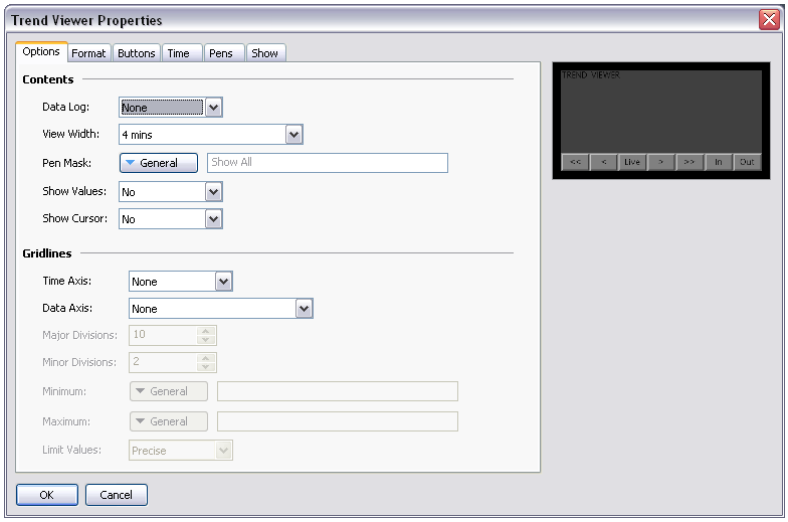


Its core properties are contained on a single tab, and all are conventional.

TREND VIEWER

The trend viewer allows the display of information from the Data Logger.

OPTIONS PROPERTIES



- The *Data Log* property selects the data log to be displayed.
- The *View Width* property is used to indicate the initial amount of data to be shown across the window. The user can subsequently zoom in and out using the buttons at the bottom of the viewer.
- The *Pen Mask* property is used to provide a 32-bit integer value to selectively enable or disable the display of specific channels. Bit 0 corresponds to the first channel of the data log, bit 1 to the second and so on. A bit value of one shows the channel while a value of zero hides it. A blank entry provides the default behavior, with all channels being displayed.
- The *Show Values* property enables or disables the display of the data values associated with each channel of the data log, either during live mode or when scrolling back and forth using the cursor.
- The *Show Cursor* property is used to enable or disable the display of a cursor on the viewer. The cursor can be activated by the user to allow a specific point in time to be precisely determined, and optionally to allow the associated historical data values to be displayed.
- The *Time Axis* property defines whether gridlines should be displayed for the time axis. The pitch of the gridlines is automatically determined by Crimson based on the amount of time covered by the viewer.
- The *Data Axis* property is used to control the display of gridlines for the data axis. Gridlines may be defined manually by specifying either just major divisions or both major and minor divisions, or may be defined automatically by specifying minimum and maximum values for the data axis and letting Crimson calculate the best gridline pattern.
- The *Major Divisions* and *Minor Divisions* properties define the number of divisions to be drawn when using manually-defined gridlines.

- The *Minimum* and *Maximum* properties are used to indicate the range of data to be displayed when using automatic gridlines. Crimson will use these values to determine the best gridline pattern to adopt. Data values will also be scaled to these values, as opposed to being scaled to their own data limits.
- The *Limit Values* property specifies how the top and bottom values of the scale are determined. If a setting of Precise is specified, the Minimum and Maximum values will be used exactly, even if this produces limits that do not exactly correspond to the automatically selected tick spacing. This can produce irregular gridline spacing. A setting of Rounded allows the scale primitive to adjust the limits to achieve regularly spaced tick marks.

FORMAT PROPERTIES

These properties are used to specify colors and fonts. Their operation is conventional.

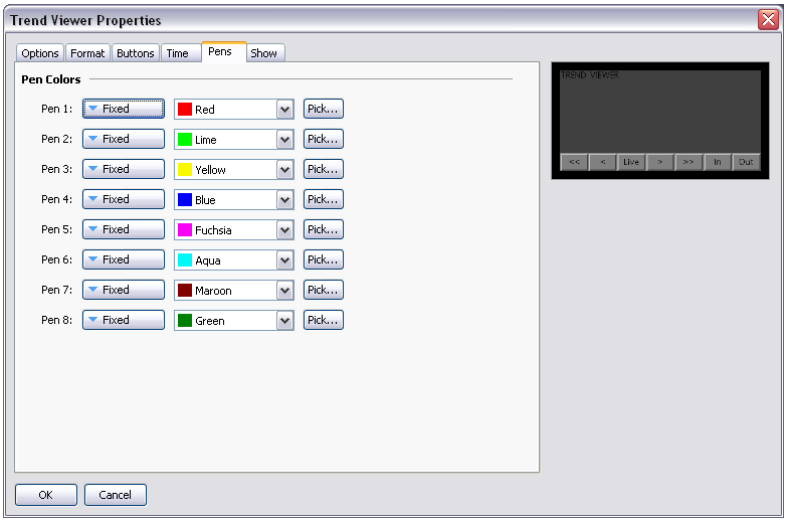
BUTTONS PROPERTIES

These properties are used to edit and optionally translate the various button labels.

TIME PROPERTIES

The Time tab is used to format the time to be used when providing time and date information relating to the data log. Refer to the chapter on Using Formats for detailed information.

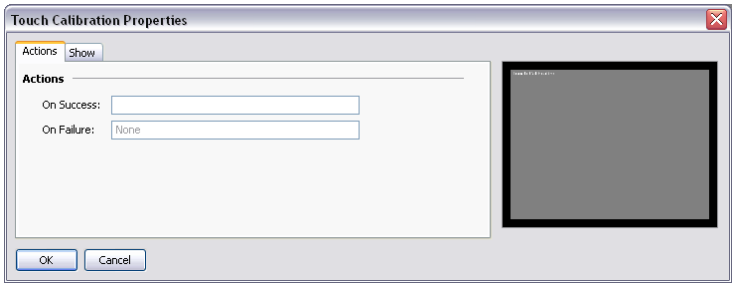
PENS PROPERTIES



These properties are used to specify eight colors that will be used for drawing the data. The colors are used cyclically, such that a ninth channel will return to the first color. Drawing so many channels is not recommended, as it can produce a very confusing display.

TOUCH CALIBRATION

The Touch Calibration primitive is used to calibrate the touch-screen…



Its primitive-specific properties define the actions to be taken when calibration has either succeeded or failed. These properties are typically configured to return to a menu screen or to otherwise move away from the calibration page.

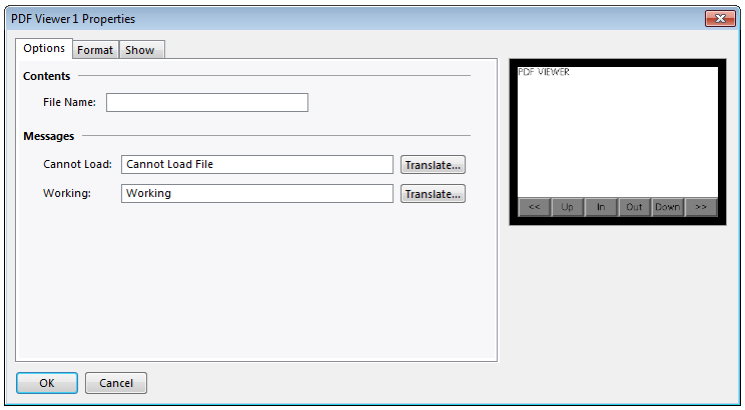
TOUCH TESTER

The Touch Tester primitive allows the user to check the touch-screen performance and calibration. Each touch produces a dot on the screen, with a trail being displayed of the last so-many touches. It has no configurable properties beyond visibility control.

PDF VIEWER

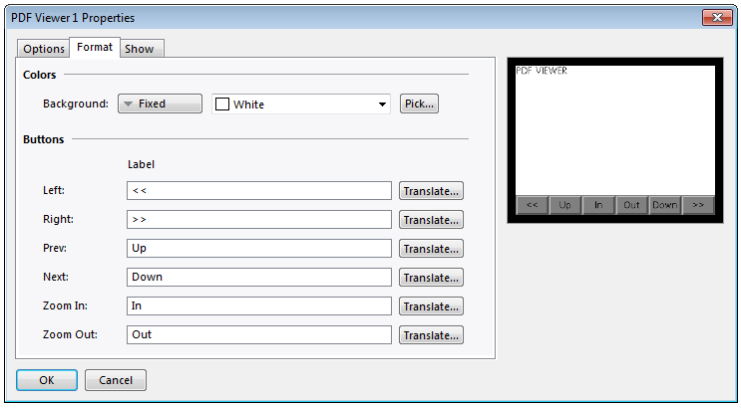
The PDF Viewer is used to display pdf files stored on the unit's CompactFlash card.

OPTION PROPERTIES



- The *File Name* property is used to specify which pdf file to display. The pdf file must be stored on the unit's CompactFlash card in a directory named "pdf."
- The *Cannot Load* property defines the text that is displayed when the named file cannot load.
- The *Working* property defines the text that is displayed when a file is being loaded.

FORMAT PROPERTIES



- The *Colors* property is used to set the background color of the viewer screen.
- The *Buttons* property is used to define the viewer button labels.

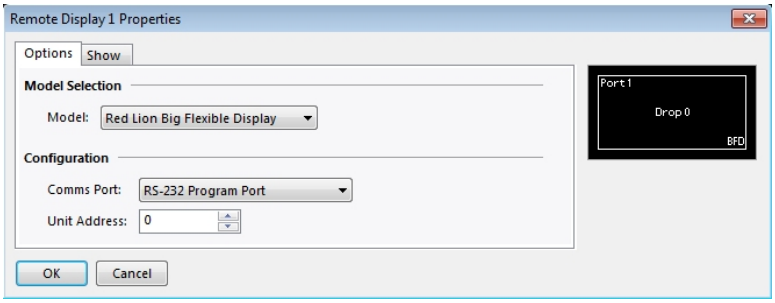
REMOTE DISPLAY

The remote display primitive is used to send part of the operator interface display to an external large display such as the PFM (Plant Floor Marque) or BFD (Big Flexible Display). This primitive sends the graphical information within its area to the slave large display. It is used in combination with the Red Lion display serial port drivers. The appropriate driver is selected in Communication.

Each pixel on the graphic area represents an LED on the remote display matrix. In the case of a color unit controlling a Big Flexible Display, which is monochrome, any colors will result in a lit LED. Only black will result in an off LED.

More than one remote display is programmable on a single screen as each can have a single unit address. The properties of the remote display are available when double clicking the primitive.

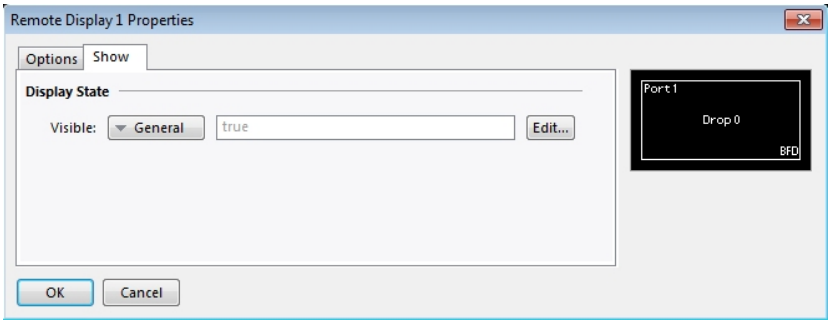
OPTION PROPERTIES



- The *Model* defines the type of slave display the graphical area will be sent to. The big flexible display, for example, will cover an area of 128 x 64 pixels on the screen.

- The *Comms Port* points to the serial communication port number where the large display for this primitive will be connected. Port numbers starts at one with the communication port.
- The *Unit Address* is the large display node address this primitive will send the information to. Please refer to the large display documentation to find the address required.

SHOW PROPERTIES

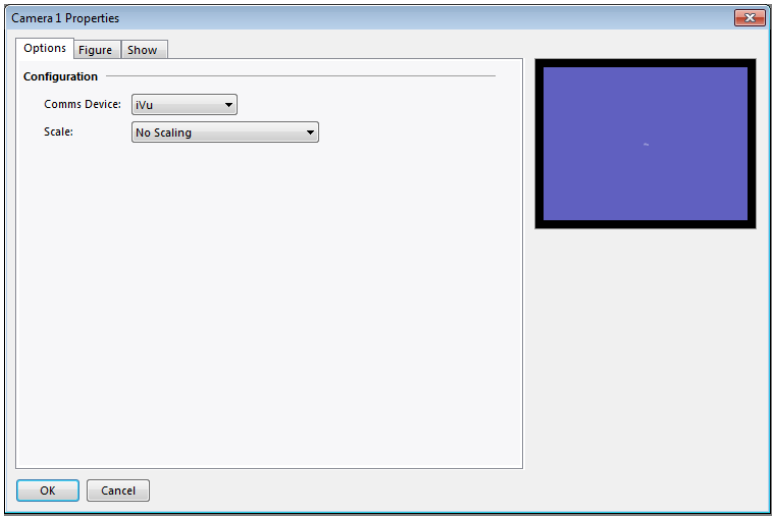


- The *Visible* setting is used to set whether to display the information in the primitive on the remote display (*true*) or not to display the information (*false*).

CAMERA

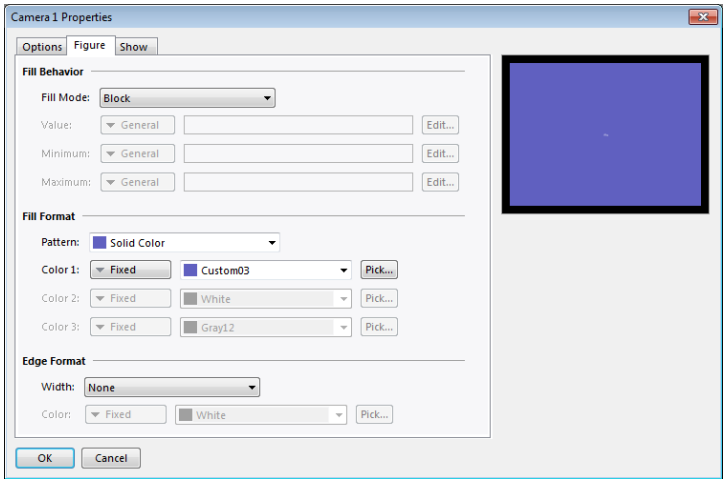
The camera primitive is used to allow video from a remote camera to be displayed on an HMI page.

OPTION PROPERTIES



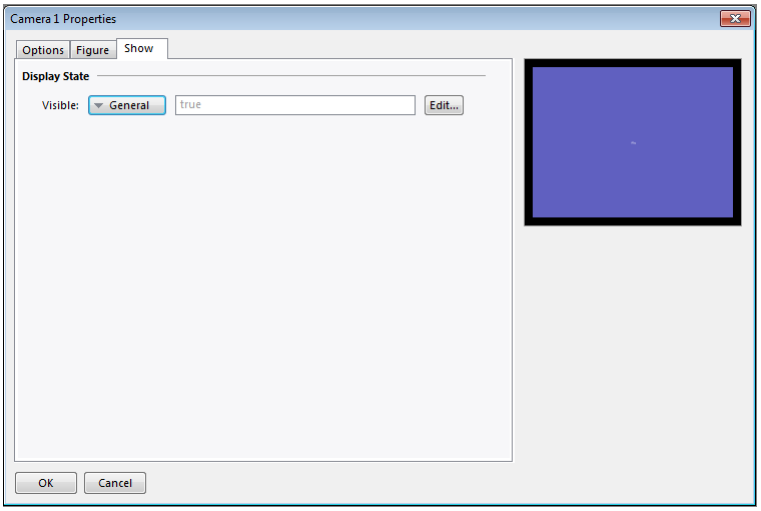
- The *Comms Device* sets the source of the video to be displayed in the given primitive. A device must already be mapped to a communications port before this selection can be made.
- *Scale* is used to change the image size coming from the source. Scaling might not be supported by all camera types.

FIGURE PROPERTIES



- The *Figure* settings are used to set the display characteristics of the primitive itself.

SHOW PROPERTIES



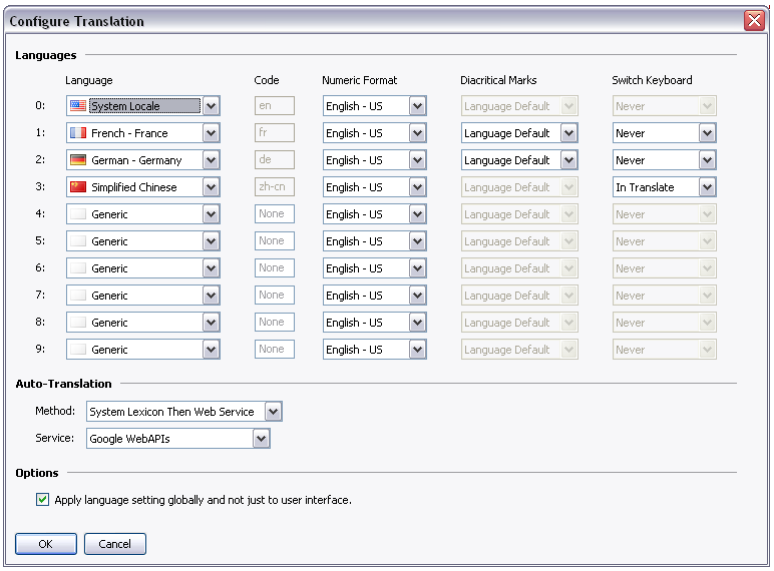
- The *Visible* setting is used to set whether to display the video in the primitive (*true*) or not to display the information (*false*).

LOCALIZATION

Crimson 3 supports a number of features that allow you to adapt your database for deployment in multilingual environments. This chapter describes how these features are used, and how you can easily create databases that can be used across the world.

SELECTING LANGUAGES

The first stage in creating a multilingual database is to configure the languages to be used within your project. Pressing the Configure Translation button on the Global page of the user interface properties displays the following dialog box…



The top section of the dialog box defines a number of properties for each language…

- The *Language* property is used to select the required language. A language may exist in several variations according to the different countries in which it is spoken. The Generic setting may be used for languages that are not directly supported within Crimson.
- The *Code* property is used to display or enter the two-character code for the language that has been selected. This property will be passed to the web translation services during automatic translation, and will be used to define the header row in a lexicon file. You must enter the code manually for Generic languages of which Crimson has no prior knowledge.
- The *Numeric Format* property is used to define whether Crimson will format numbers using US format or using a format specific to the current language selection. Numeric formatting options include the use of commas versus decimal points, and the placement of digit grouping characters.
- The *Diacritical Marks* property is used to override a language's default setting for the treatment of accents on upper case characters. For example, French as used in France (as opposed to Canada) applies accents to upper case characters, which can make these characters

harder to render in some fonts. Selecting Lower Case Only for this setting will override this default behavior.

- The *Switch Keyboard* property is used to select the circumstances in which the Crimson configuration software will switch the keyboard layout to that used by the language. The switching can occur when using the translation dialog box, whenever text is being edited in this language, or not at all. Keyboard switching in the translation dialog is enabled by default for languages such as Simplified Chinese, thereby ensuring that the appropriate Input Method Editor is invoked.

The next section controls auto-translation, and is described below. The final property selects whether the current language setting is applied to services such as the web server and the data logger, or whether these should always use the system default language.

CONFIGURING AUTO-TRANSLATION

Crimson contains powerful auto-translation features to help you adapt your database for international deployment. Auto-translation comprises two components, namely a system lexicon and a web-based translation service.

The system lexicon is a Unicode text file that contains many standard words and phrases that are used in industrial automation and process control, together with translations in each of a number of common languages. This lexicon can be consulted during the translation process, allowing commonly occurring-text to be translated very quickly and with a high degree of accuracy.

The web-based portion uses one of two services. Google WebAPIs typically provides faster translations, as it is not subject to bandwidth restrictions. In contrast, Microsoft Translator provides more accurate translations, but is not as quick as there is a limit on the number of submissions that can be made per minute. You can select either service from the Configure Translation dialog pictured above.

Auto-translation can be configured to use either or both of these methods. If you have an Internet connection, it is generally better to use first the lexicon and then the web-based service. The lexicon can be used alone in some circumstances to avoid the questionable translations that the web-based services can sometimes provide.

TRANSLATING YOUR DATABASE

Database translation can be accomplished in a number of ways.

ENTERING TRANSLATIONS

Manual translation is performed by pressing the Translate button next to each translatable string in the database. A dialog will be displayed, allowing translated text to be entered, or allowing auto-translation to be invoked for this string alone...

Local auto-translation like this allows you to review the translations for accuracy.

GLOBAL AUTO-TRANSLATION

The Utilities submenu on the File menu contains a command to apply auto-translation to every string in the database. This command may take some time to execute, especially if a bandwidth limited translation service is used. Some care should be taken when using global auto-translation, as strings that are not in the system lexicon may be subject to erroneous translation if they contain technical terms or industry-specific jargon.

EXPORTING AND IMPORTING

The Utilities submenu also contains commands to export and import text files containing all the translatable text in the database. These files can be edited using an application such as Microsoft Excel, allowing translations to be entered manually. This facility is particularly useful when working with a third-party translation service. The file format includes several columns that indicate the source of each string, allowing distinct occurrences of a given string to be translated into different text according to context.

APPLYING A LEXICON

In addition to the system lexicon described above, you may create your own lexicons, either from scratch or by using Export Lexicon command in the Utilities submenu. Lexicon files are Unicode text files that start with a header row containing tab-separated language codes, as used in the Code properties defined in the Configure Translation dialog box. After the header row, each row contains a word or phrase in each of the defined languages.

A sample lexicon file is shown below…

```
en          fr          de
one         un          eins
two         deux        zwei
three       trois       drie
```

Note that text should be entered in lower case unless a specific term is only ever used in upper case, such as might be the case with a German noun. The use of lower case allows Crimson to form its own upper case and title case variants.

PREVIEWING TRANSLATIONS

Translations can be previewed within the graphics editor by selecting the appropriate language from the drop-down menu that is accessed via the flag icon in the toolbar. Any direct editing of text will also apply to the currently selected language, with the other languages being left unchanged. Editing within dialog boxes continues to be restricted to the default language, with the other languages accessed via the Translate button as usual.

SWITCHING LANGUAGES

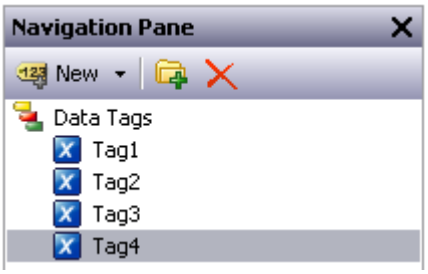
The language used by the target device is controlled via calls to the `SetLanguage()` function, with the argument of the function being a number between 0 and 9 to select the required option. For example, a call to `SetLanguage(1)` in the example above will select French, while a custom action of `SetLanguage(2)` will select German. The `GetLanguage()` function can be used to determine the current language.

USING WIDGETS

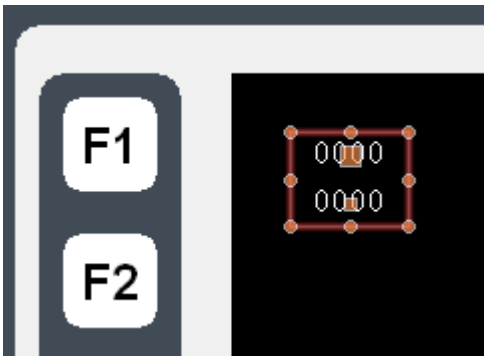
Crimson 3 supports a powerful new feature—the ability to turn ordinary groups of primitives into powerful entities called widgets. In addition to its component primitives, a widget contains user-definable data items that can be edited at the group level but referenced by the widget's components. This chapter explains how to create widgets, and how to use them.

CREATING A WIDGET

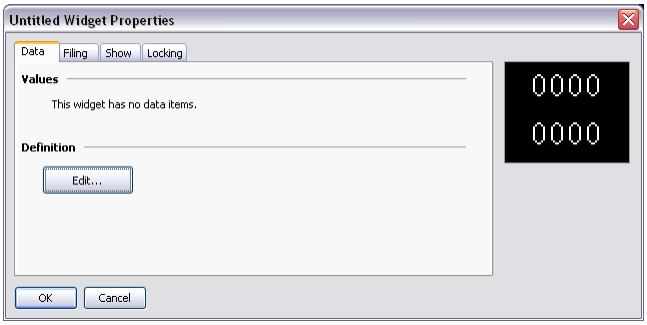
The easiest way to understand widgets is to create one. Let's start by creating an empty database and adding four numeric tags. Leave the tag properties at their default settings, resulting in four internal integer values named Tag1 through Tag4.



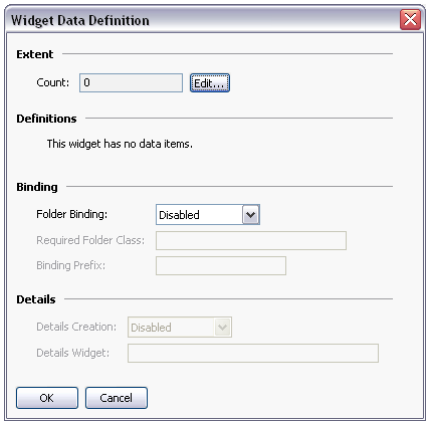
Switch to the Display Pages section and add two data box primitives to the page...



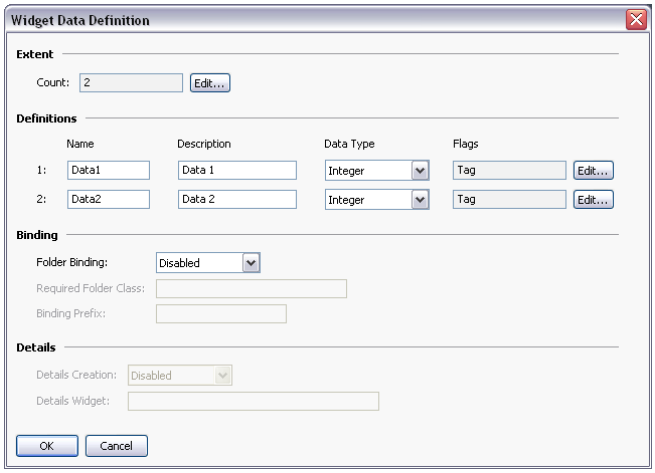
Leave their properties at their default values for now, and select both items. Right-click on the selection, and select the wonderfully-named Widgetize command from the context menu. The items will be bound into a group, but the following dialog box will also appear…



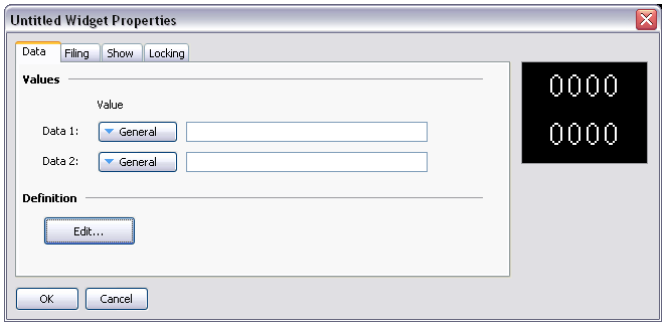
Once the widget has been created, this dialog box will be used to edit the widget's data items, but for now we have nothing defined. Click on the Edit button in the Definitions section to allow us to define some data items…



Clicking on the Edit button next to Count field will let us create two properties…

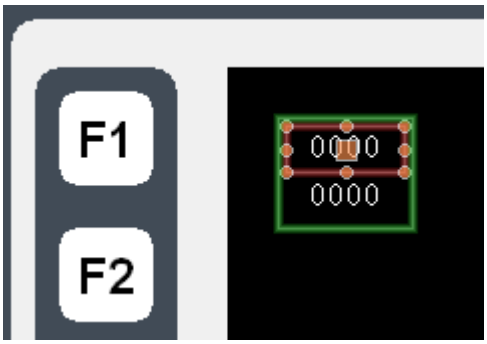


Complete the data fields as shown above, paying particular attention to get the data type right, and to modify the Flags fields to indicate that each data item should be a tag. (The flags field can be edited using the Edit button next to the property.) Press OK to close the dialog box, and note how the widget itself now displays data items in its own properties dialog…

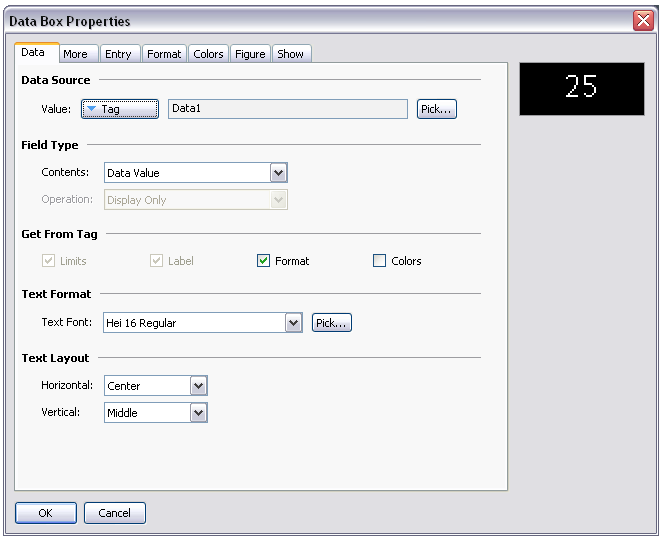


Ignore these for now, and press OK to close this dialog, too.

The widget should still be selected in the graphic editor, so click on one of the data boxes contained in the widget to enter group editing mode. Remember, the green rectangle marks the group that we're editing, and the red rectangle shows the selected item in that group…



Double-click on the data box to bring up its properties…

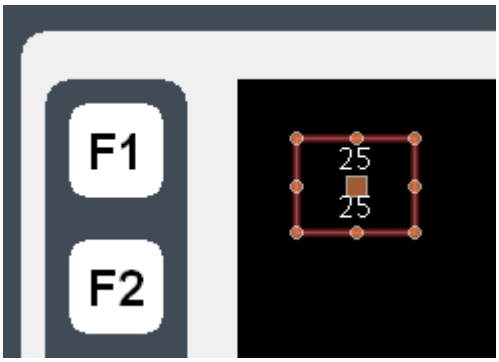


Enter `Data1` in the Value field, and note the results.

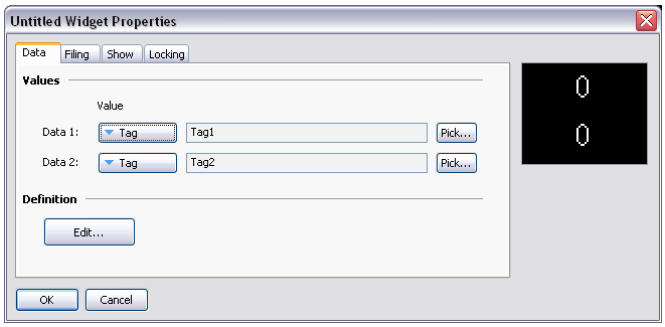
Crimson accepts this as a tag name, even though we don't actually have a tag called Data1 in our database. This value is actually equal to one of the data items defined within the widget, and will represent whatever tag we assign when we go back in and edit the widget data. (The value of 25 shown in the preview window is the default value used for widget data items that are not mapped to anything.) Since Data1 is marked a tag, we can access its properties, use it as a source of formatting information, or do anything else that we would do with a tag.

Repeat this step for the second data box, this time setting its Value property to `Data2`.

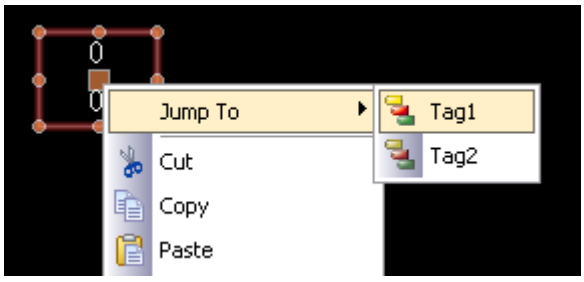
Press **Esc** until you have the widget alone selected. If you go too far and clear the selection, just click on the widget itself, ensuring that it has a red rectangle round it…



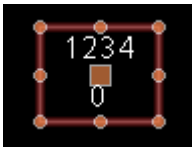
Now bring up the widget properties, and this time enter values for the data items…



Enter the values shown above, setting the data items to `Tag1` and `Tag2` respectively. Note how the preview now shows values of zero, as the data boxes within the widget are now getting their data from Tag1 and Tag2 respectively. To make things a bit more interesting, right-click on the widget and access the Jump menu…

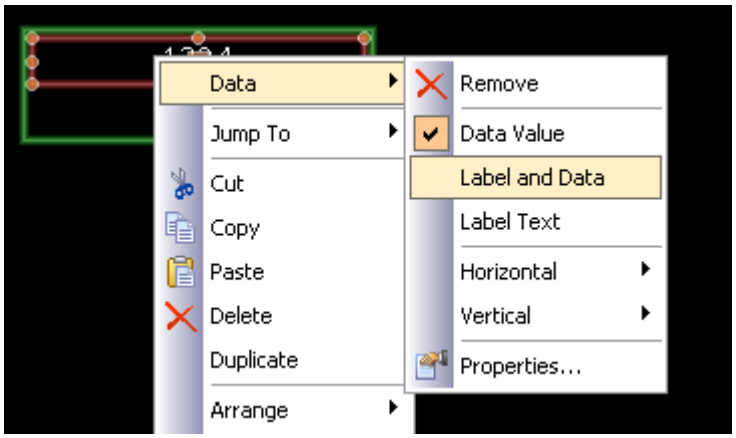


Select Tag1 to jump to that tag, and enter a value of 1234 in the Simulate As property. Use the **ALT+LEFT** key combination or the Back button on the toolbar, and note how the widget is continuing to track the tag data…

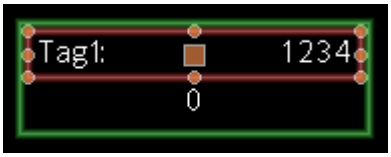


Next, grab the right handle and make the widget a little wider…

Left-click on the upper of the two data boxes to enter group editing mode, and then right-click on the same box to access its context menu…



Select the Data submenu, and choose the Label and Data command to configure this data box to display the tag's label as well as its data value. Note the new appearance of the widget…



As you can see, the data box is displaying the label from Tag1, indicating that the value of `Data1` that we entered into the data box's Value property is entirely equivalent to the tag to which the data item was subsequently configured. We refer to the process of setting a widget item to a tag as binding that data item to that tag. Binding can be performed in more complex ways, as we shall see later.

SUMMARY

Let us now recap what we did…

- We placed primitives on the display and grouped them into a special kind of group called a widget. The widget appeared to behave like a normal group in terms of editing and so on, but had additional properties.
- We edited the data definitions for the widget, creating two data items. Each was given a name, a description, a data type and a number of flags.
- We used group editing to edit the contents of the widget, setting their properties to the widget's own data items, referring to them by the data item names.

- We modified the widget's data items, binding them to tags, thereby providing real tags and their associated information to the contents of our widget.

WHY THIS MATTERS

So why are widgets important? We could easily have created the data boxes and bound them directly to the tags, so why bother with these extra steps? The answers become obvious when you try to create more complex widgets…

- Widgets allow data items be used in several places, with multiple elements in the widget being dependent on the same tag without your having to select the tag name in multiple places.
- Widgets can encapsulate complex design and functionality, and allow you to replicate and reuse this across or within databases. In effect, they allow complex primitives to be created by the user.
- Widgets can be saved to disk and be added to the Resource Pane, or distributed via email, thereby allowing easier cooperation between Crimson users or between users and Tech Support.

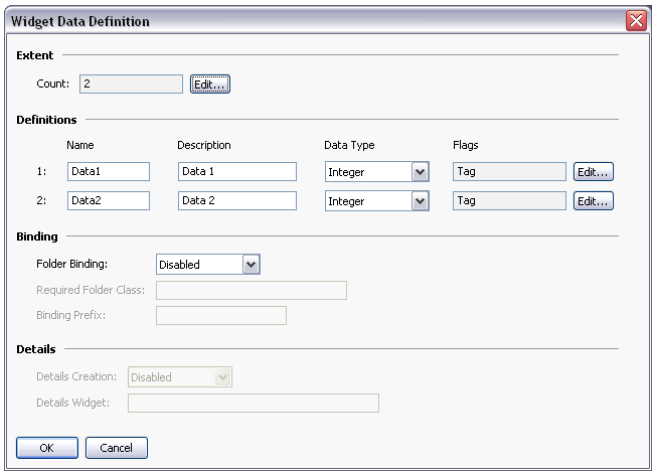
DOWN TO DETAILS

The next section revisits most of the topics above, but in more detail.

They also explore some of the magic that can be used to make widgets even more powerful.

WIDGET DATA DEFINITIONS

The features that give widgets their power is their data items. The data definition of a widget is edited by opening the widget's properties and by clicking on the Edit button in the Definitions section of the Data page…



- The *Extent* property defines how many data items are required for this widget. This value can be changed at any time, but making it smaller will result in data items and their values being lost. Up to eighty data items may be defined.

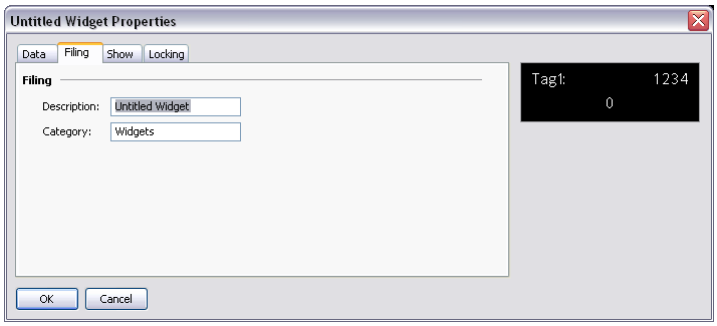
- The *Name* property of each data item is used to refer to that item from primitives contained within the widget. It must therefore meet all the requirements of a tag name. It must contain no spaces or punctuation, and it must start with a letter.
- The *Description* property for each data item is used to provide a more friendly version of the name, this time for display in the data item editing dialog. No restrictions are placed on the contents of this field.
- The *Data Type* property for each item defines the required data type. The way in which the data item is displayed in the widget's property dialog will depend on the setting that is selected. The real, integer and string data types correspond to expression values, while the color, page and action data types allow more complex items to be created. Page and action items can be treated as display page names and programs from within the widget's primitives.
- The *Flags* property for each data items is used to modify items that have data types of real, integer or string. It supports the following settings...

SETTING	DESCRIPTION
Tag	The value entered for the data item must be a tag. The primitives within the widget can treat the data item as a tag, and access its properties, data format and so on.
Writable	The value entered for the data item must writable. The primitives within the widget are similarly allowed to write to the data item.
Array	The value entered for the data item must be the name of an array. The primitives within the widget will see the data item as an array, and must use the index operator to access individual values.
Element	The value entered for the data item must be an array element. The primitives within the widget will see the data item as an element, and will be able to pass it to functions that require arguments of this type.
No Bind	Crimson will not apply folder binding to this property, allowing it to be used to store predefined values without errors being generated on binding. See later sections for details of folder binding.

- The *Binding* property group is used to control an advanced feature known as folder binding. It is discussed in detail below.
- The *Details* property group is used to control an advanced feature known as details page creation. It is discussed in detail below.

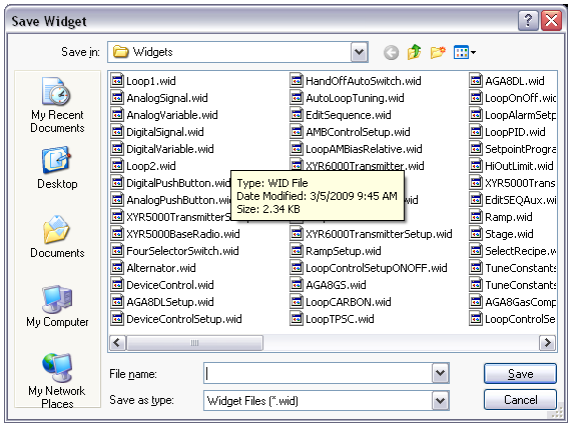
FILING WIDGETS

Each widget has a Filing tab in its property dialog…



The *Description* and *Category* properties are used to control how a widget will be displayed on the Resource Pane after it is saved. All widgets of the same category will be grouped together in the same subcategory on the Primitives category, and the widget description will be displayed when the user hovers over an item.

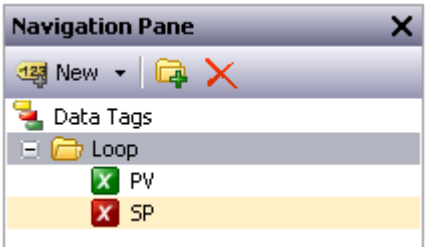
To save a widget, simply select it, and choose Save Widget from the Edit menu or press the **CTRL+Q** key combination. A standard Save dialog will open, allowing you to save the widget as a `wid` file in the Crimson widget directory…



The Resource Pane will update automatically whenever a widget file is added to this directory. This will occur whether the change is made via Crimson or by simply dropping a `wid` file in the directory via Windows Explorer. Note that widget files are stand-alone, and can be transferred between Crimson installations on different machines. This provides a powerful mechanism for sharing user interface elements, or for exchanging items with other engineers when multiple individuals are working on a project.

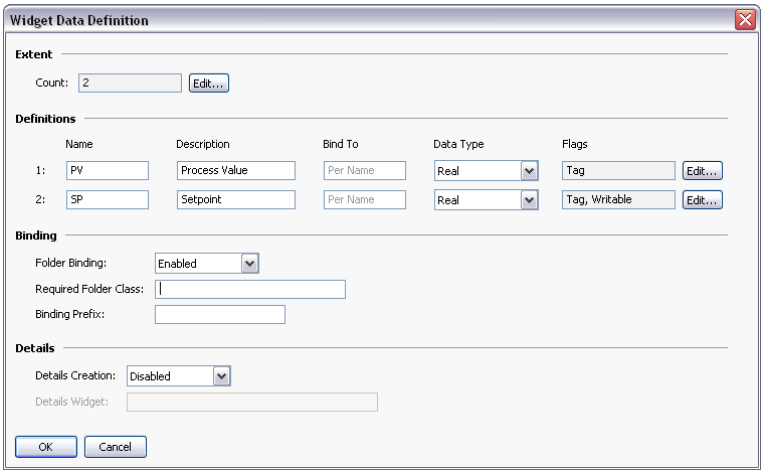
FOLDER BINDING

Crimson's ability to organize tags in folders allows a kind of object-oriented design whereby tags that represent the properties of an object can be grouped together in a folder that represents the object itself. Consider the example below…



Here, a folder has been created to represent a PID loop, and tags have been created to refer to the loop's process value and setpoint. The tags are referred to in code as `Loop.PV` and `Loop.SP`, using the standard Crimson rules for using nested items.

Folder binding allows you to create a widget that mirrors that object and property structure that you have created in your tags. Consider the following data definition…



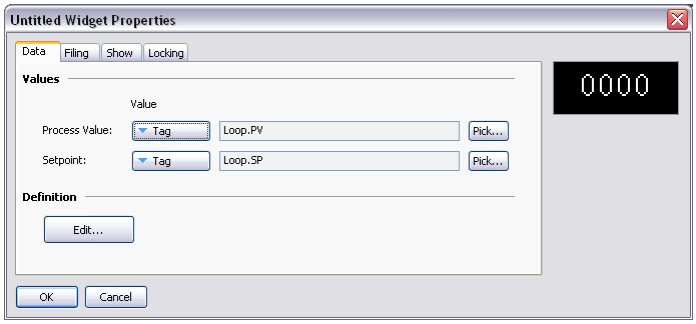
Here we have created data items whose names match the names of the tags that make up a PID loop. We have provided human-readable names for them, and we have flagged both data items as being tags. We have also defined the setpoint to be writable. Note at this time that a new property called *Bind To* has appeared for each data item—we shall return to this during a discussion of advanced folder binding.

In the binding section, we have enabled folder binding. This indicates that we want Crimson to support the automatic binding of all the data items to tags from a single source folder. After we save these changes and select the widget's context menu, we will notice a new command called Bind Widget that allows the binding operation to be performed.

Selecting the command or pressing **CTRL+B** displays the following dialog box…

If we drag the Loop folder from the Resource Pane and drop it on the target, the widget's data items will be automatically bound to the corresponding tags in the folder.

Opening the widget's properties shows the results…



In other words, each data item has been bound to the tag within the selected folder that has a name equal to its own data item name. Think for a second about how powerful this is: You can define multiple properties and bind them in a single operation, reducing design time and allowing better reuse of previously designed items.

ADVANCED BINDING

Folder binding supports a number of advanced options.

CLASS MATCHING

The first and simplest is the *Required Folder Class* setting in the widget's properties. This can be used to restrict the folders that will be accepted during binding, therefore avoiding mismatches between what amount to different object types. The specified class on the widget must match the class on the folder that is being bound, or an error will result.

BINDING PREFIXES

The *Binding Prefix* property can be used when nesting widgets to allow the child widgets to be bound to sub-folders of the folder to which the parent widget is bound. For example, suppose you create a dual-loop widget that is to be bound to a folder that contains two PID folders named Loop1 and Loop2. By setting each of the child widget's binding prefix to one of the loop names, you can ensure that they are bound to different child folders of the folder that is dragged on to the parent widget. For example, if the first child widget has a binding

prefix of `Loop1` and its parent is bound to a folder called Dual, the child widget's properties will be bound to expressions of `Dual.Loop1.PV` and `Dual.Loop1.SP` respectively.

USING BIND TO

The *Bind To* property of a data item can be used to modify the expression to which that data item is bound. The simplest option is to enter a name distinct from the name of the data item, in which case that name will be used for selecting the tag to which to bind.

USING PERIODS

You may also enter a name that contains periods. These select tags in child folders of the source folder. For example, entering `Remote.SP` will result in the data item in question being bound to an expression of `Loop.Remote.SP` upon binding to the Loop folder.

USING CARETS

To ascend the folder tree, you may prefix the name with caret characters, each of which moves up one level. A data item with a Bind To setting of `^Name` in a widget that is bound to a `Dual.Loop` will itself be bound to the expression of `Dual.Name`.

SPECIAL NAME

You may also use one of a number of special *Bind To* names…

NAME	RESULT
`::Path`	The full path of the tag to which this widget was bound, including any parent folders.
`::Name`	The name of the tag to which this widget was bound, excluding any parent folders.
`::TopPath`	The full path of the tag to which the root widget was bound in a nested binding operation. Equivalent to `::Path` for non-nested binding.
`::TopName`	The name of the tag to which the root widget was bound in a nested binding operation. Equivalent to `::Name` for non-nested binding.

Note that each of these special names evaluates to a string constant equal to the required name and not to the actual tag itself. They are typically used to provide information to the user regarding the folder to which a widget or its root widget have been bound.

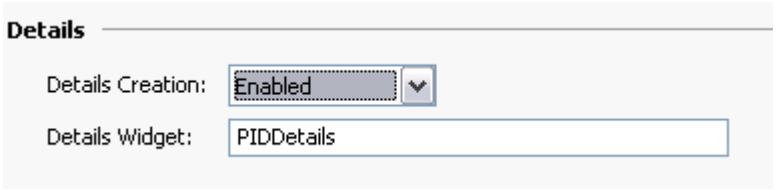
DETAILS WIDGETS

Suppose you have created a PID widget, but want to display more detailed status information when the user presses a button in that widget. The easy answer is to create a more and perhaps larger complex widget that you would then bind to the same loop. You would place this widget on another page, and then select that page from the original overview widget, perhaps using a data item to tell the widget which page to use.

Well, Details Widget creation performs all these steps automatically!

ENABLING DETAILS CREATION

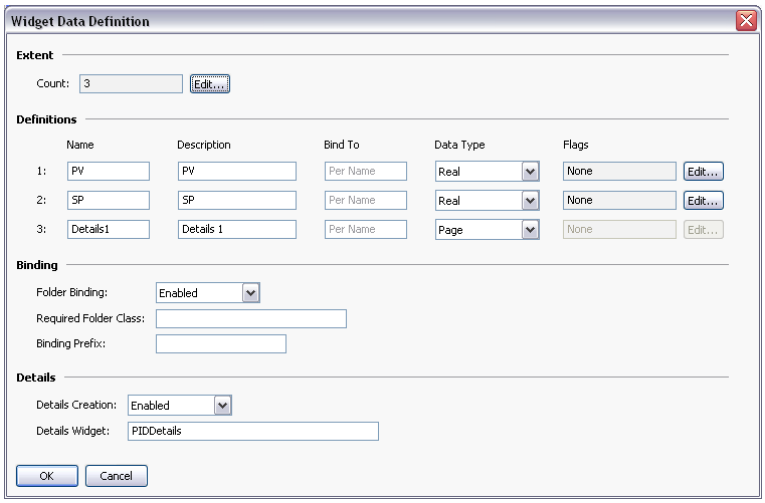
This feature is controlled via the *Details Creation* property of the widget's data definition…



The *Details Widget* property is used to provide a comma-separated list of the one or more details widgets that you would like to place on their own pages. Each widget is specified by giving the filename to which it was saved. In the example above, we have one details widget to be extracted from a file called `PIDDetails.wid` in the Crimson widgets directory.

DEFINING DATA ITEMS

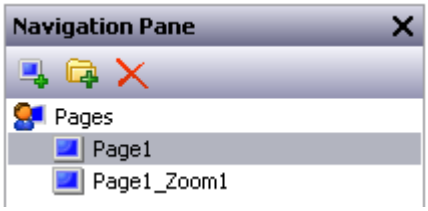
We must also provide data items in the overview widget so that we can access the names of the pages that are created for the details widgets. These properties must be named `Details1`, `Details2` and so on, with one data item for each element in the Details Widgets list. Each data item must be the Page data type. In the example below, we have created a single such property to hold the page name of our single details page…



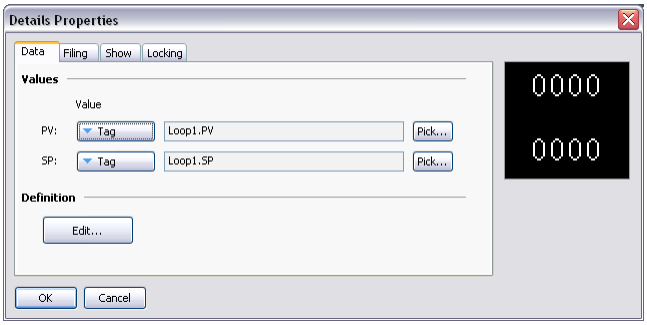
RESULTS OF BINDING

When we bind the overview widget to our PID loop, a new page is created to hold the details widget. The name of the new page is based on the name of our page containing the overview widget, but with a "Zoom" suffix and a number chosen to make the name unique.

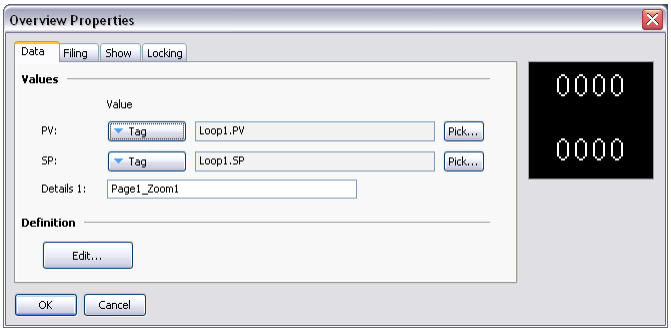
This page is placed in the Navigation List below the current page…



The details created on this page is bound to our loop…



And the properties of our overview widget are modified as follows…



We can easily define a button within our overview widget, and have this button invoke an action of `ShowPopup(Details1)` thereby displaying the associated details widget. The details widget itself can close the popup by calling `HidePopup()`.

MULTIPLE DETAILS PAGES

If multiple details pages are created, you will recall that data items called `Details1`, `Detail2` and so on in the overview widget will hold the names of those pages. These data items can also be defined on the details widgets, and will also be set to the names of the pages that have been created. This is useful if you want to allow the first details page to navigate to the second and so on, thereby linking the pages together. Details widgets can also define a special data item called `DetailsP` which will be set equal to the page that holds the overview widget. This can be used to return to the overview—something that cannot be achieved via a simple `GotoPrevious()` when multiple details pages are provided.

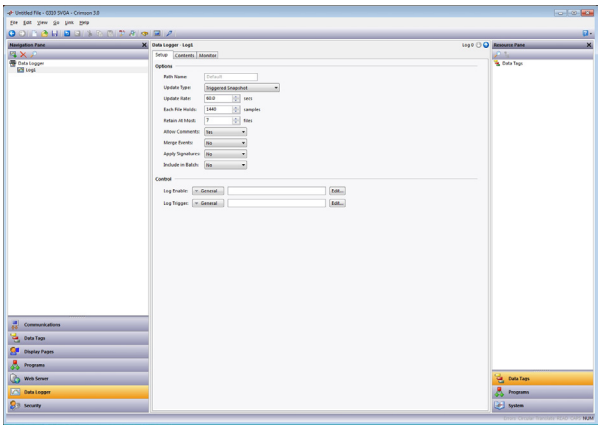
USING THE DATA LOGGER

Now that you have configured the core of your application, you may decide to make use of Crimson's data logger to record certain tag values to CompactFlash. Data recorded in this way is stored in industry-standard comma-separated variable or CSV files, and can easily be imported into applications such as Excel using a variety of methods. To configure data logging, select the Data Logger category in the Navigation Pane.

CREATING DATA LOGS

Data logs are created in the Navigation List in the usual way.

Each log has the following properties…



SETUP PROPERTIES

- The *Path Name* property allows you to modify the directory in which the log will be saved. By default, the log is saved in a directory named after the log's own name. This property allows you to rename logs in a way that is not compatible with the 8.3 filename convention while still using valid logging directories.
- The *Update Type* property allows the user to choose between *Continuous Sample* and *Triggered Snapshot* data collection.
- The *Update Rate* property is used to indicate how often Crimson will take a sample of the data items to be logged. Although a decimal place can be entered, sampling is only accurate to 200ms. The fastest sample rate is one second, but note that using such a high rate will produce very large amounts of data. All of the tags in the log will be sampled at the same rate.
- The *Each File Holds* property is used in conjunction with the *Update Rate* property to determine how often a new data file will be created. When in *Continuous Sample* mode, this is the number of samples that will be included in each log file. Typically, this value is set such that each log file contains a sensible amount of data. For example, the log shown above is configured to use a new log file each day. When in *Triggered Snapshot* mode, each file will only contain as many samples as were logged as a result of the given trigger within the time frame between new log files. Each new log file is given a different name.

- The *Retain At Most* property is used to indicate how many log files will be kept on CompactFlash before the oldest file is deleted. This property should be set so as to allow whatever is consuming the logged information to extract the data from the Crimson device before the information is deleted. The log shown above is configured to retain a week's worth of data.
- The *Allow Comments* property is used to enable or disable the addition of comments to the data log via the `LogComment()` function. Refer to the Reference Manual for details of how this function can be used.
- The *Merge Events* property is active when the *Allow Comments* property is set to *Yes*. The *Merge Events* property allows event comments to be merged into the data log. Events will still be logged in the events file.
- The *Apply Signatures* property is used to control the addition of cryptographic signatures to the log. See below for more information on these signatures and how they can be used to ensure the log integrity is maintained.
- The *Include in Batch* property is used to include or exclude this log from the batch logging system. See below for information on how batch logging operates.
- The *Log Enable* property is used to allow or inhibit logging. If the entered expression is true, logging will be enabled. If the expression is false, logging will be disabled. If no expression is entered, logging will be enabled by default.
- The *Log Trigger* property is active when the *Update Type* property is set to *Triggered Snapshot*. The *Log Trigger* property is used to set the event that initiates taking a snapshot of current data points.

CONTENTS PROPERTIES

- The *Contents* property is used to indicate which tags should be included in the data log. Tags can be dragged into the list from the Resource Pane, and moved up and down within the list using standard drag-and-drop techniques.

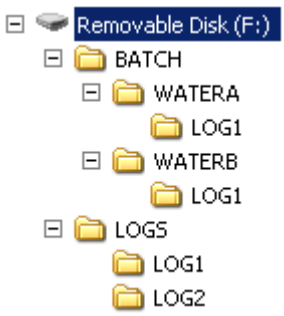
MONITOR PROPERTIES

- The *Monitor* property is active when both the *Allow Comments* and *Merge Events* properties are set to *Yes*. The *Monitor* property is used to indicate which tags will be monitored for event comment merging into the data log.

BATCH LOGGING

When you first access the Data Logger, you will notice a global setting to enable or disable batch logging. For normal data logging operation, the Data Logger will save the log files under the folder name specified for each log. Batch logging, on the other hand, also saves all logs that are so configured to a directory named after the current production batch. This allows all the logs related to a particular batch to be accessed and manipulated as a group.

To illustrate this, consider the following directory structure…



This example is taken from a target device that has batch logging enabled and has two data logs configured. The first data log is set to be included in the batch, while the second one is not. Note that the log files are stored by default in the directories named `LOGS/LOG1` and `LOGS/LOG2`. Note also, however, that the first log is also being placed in subdirectories under the `BATCH` directory. Each subdirectory contains the data sampled between the time when that batch was started and the time when the batch was ended.

CONTROLLING A BATCH

Batch logging is controlled via a number of functions. `NewBatch(`*name*`)` will create a folder called *name*, ending the current batch and starting a new one. Files recorded after this command will be saved under the new folder. The `EndBatch()` function will stop the current batch, while `GetBatch()` will return the name of the batch that is currently active. For more information, please refer to the Reference Manual.

DIGITAL SIGNATURES

Crimson supports the addition of cryptographic signatures to data, event and security logs, thereby allowing you to check a log file's integrity. The signatures will show if the log has been tampered with, or if data has been removed from the middle of the file. They will also allow you to be sure that a given log file came from the given device.

Enabling signatures adds an extra field to the CSV file to allow the storage of the required data. If you examine such a file, you will see that every few lines of the file has a large amount of data stored in this additional field. This data is the digital signature—a value mathematically derived from the data in the proceeding lines in a manner that makes it impossible to figure out how to keep the signature valid if a change is made to the data.

Signatures can be verified using the SigCheck command-line utility provided with the Crimson software. The utility will either confirm that the file is valid, or indicate the line at which the validation error occurs. If you want to be 100% sure of a file's integrity, you should also validate the digital signature applied to the SigCheck utility to ensure that it is approved Red Lion software and not a modified version that will produce invalid results.

To provide further comfort as to the integrity of the signature process, a Technical Note describing the algorithm in detail can be obtained from Red Lion support. You may also obtain the source code for SigCheck so that you can independently verify that it correctly validates the signatures applied to the log files.

LOG FILE STORAGE

As described above, data logs store their data in a series of files on the target device's CompactFlash card. The files are placed in the subdirectory specified in the log's properties, with this directory being stored under a root directory entry called `LOGS`.

Log files are named after the time and date at which the log is scheduled to begin. If each file contains an hour or more of information, the files will be named `YYMMDDhh.CSV`, where `YY` represents the year of the file, `MM` represents the month, `DD` represents the date, and `hh` represents the hour. If each file contains less than one hour of information, the files will instead be named `MMDDhhmm.CSV`, with the initial six characters as described above, and the trailing `mm` representing the minute at which the log began. These rules ensure that each log file has a unique name, dependant on the time at which is was created.

The length of each file depends on the *Update Rate* and *Each File Holds* properties. For example, with an update rate of 5 seconds and a number of samples of 360, each file will hold (5 x 360) / 60 = 30 minutes of data, therefore use the `MMDDhhmm.CSV` filename format. A new file will be created every 30 minutes, either on the hour or at half-past the hour.

THE LOGGING PROCESS

Crimson's data logger operates using two separate processes. The first samples each data point at the rate specified by the properties of each log, and places the data in a buffer within the RAM of the target device. The second process executes every two minutes, and writes the data from memory to the CompactFlash card.

This structure has several advantages…

- Writes to the CompactFlash card are guaranteed to begin only on a two-minute boundary—that is, at exactly 2, 4 or 6 minutes past the hour, and so on. This means that if your target device supports hot-swapping of CF cards, you can wait for the next burst of writes to start, and, when the CompactFlash activity LED ceases to flicker, you are guaranteed to have at least until the start of the next two-minute interval before further writes will be attempted. This means that you can remove the card without fear of data corruption. As long as you insert a new card before four minutes have elapsed, no data will be lost.
- Writes to the CompactFlash achieve a much higher level of performance, by avoiding the need to continually update the card's file system data structures for every single sample. For logs configured to sample at very high data rates, the bandwidth of a typical CompactFlash card would not allow data to be written reliably in the absence of such a buffering process.

Note that because data is not committed to CompactFlash for up to two minutes, up to this amount of log data may be lost when the terminal is powered-down. Further, if the target device is powered-down while a write is in progress, the CompactFlash card may be corrupted. To ensure that such corruption is not permanent, Crimson uses a journaling system that caches writes to additional non-volatile memory within the terminal. If the device detects that a write was interrupted during power-down, the write will be repeated when power is reapplied, thereby reversing any corruption and repairing the CompactFlash card.

If you want to remove a CompactFlash card from a panel performing data logging, you must observe the procedure described above with respect to the activity LED, and only remove power when the activity has ceased. If you are not sure if the terminal was powered-down correctly, reapply power, allow a CompactFlash write sequence to complete, and power down according to the correct procedure. The card can then be removed safely.

Since the gyrations required to remove a CompactFlash card are somewhat complex, Crimson provides a variety of other mechanisms for accessing log files, thereby eliminating the need for such removals. These methods are described below.

ACCESSING LOG FILES

There are five methods of accessing log files…

- You can mount the CompactFlash card as a drive on a PC via the process described at the start of this manual. This will allow the logs to be copied via Windows Explorer. This method has some drawbacks in terms of the amount of load that Windows can put on the CompactFlash card when it is first mounted.
- You can use the Web Server that is described in the next chapter. With the web server enabled, log files can be accessed over a TCP/IP connection, using either a web browser, such as Microsoft Internet Explorer, or by using the automated process implemented by the WebSync utility provided with Crimson.
- You can use the FTP Server to allow remote clients to connect to the Crimson device and download the logs. Refer to the Using Services chapter for details.
- You can use the Sync Manager to push the files to an FTP server on a periodic basis. Again, refer to the Using Services chapter for more details.
- You can enable automatic copying of the log files to a USB memory device by configuring the Memory Stick option in the Communication category. Refer to the Using Communications chapter of this manual for more details.

USING THE WEB SERVER

Crimson's web server can be used to expose various data via TCP/IP connections, using either modems or the target device's Ethernet ports. This allows remote access to diagnostic information or to the values recorded by the Data Logger. The web server is configured by selecting the Web Server category in the Navigation Pane.

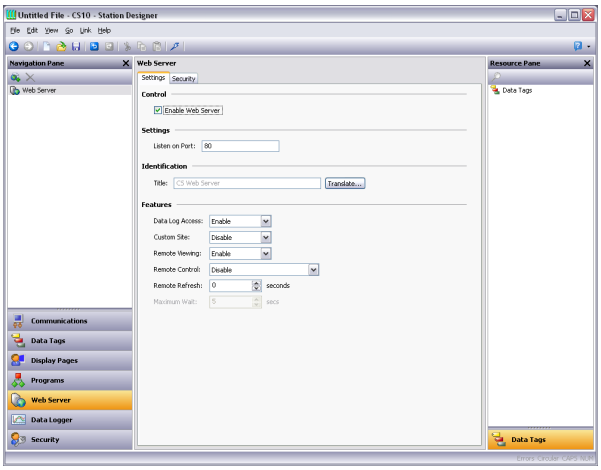
IMPORTANT NOTE

While Crimson provides a variety of protection mechanisms to limit access to the web server, you should use good engineering practices when designing your system. This means that you should avoid performing any safety-related operations via the web server, and you should ideally use an external firewall to prevent unauthorized access in case Crimson's own security protections are breached. Security is ultimately your own responsibility, and Red Lion Controls does not recommend that you rely solely on Crimson's own security measures.

WEB SERVER PROPERTIES

The web server's properties are accessed from the root entry of the Navigation List.

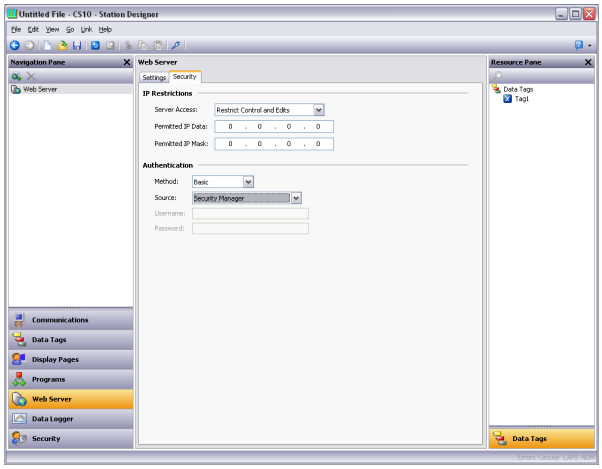
SETTING PROPERTIES



- The *Enable Web Server* property is used to enable or disable the web server. If the server is enabled, the panel will wait for incoming requests and then fulfill the requests as required. If the server is disabled, connections to this port will be refused. Remember that in order for the server to operate, a TCP/IP connection must have been configured using the Communications category.
- The *Listen on Port* property indicates the TCP port number the web server will listen on. Port 80 is the standard port used by the HTTP protocol and will most likely suit your application.
- The *Title* property is used to provide the title to be shown on the web server menu. This title can be used to differentiate between several terminals on a network, thereby ensuring that the correct terminal is being accessed.

- The *Data Log Access* property is used to enable or disable web access to the files created by the Data Logger. This facility must be enabled if the web server is to be used by a remote client program to automatically synchronize data logs.
- The *Remote Viewing* property is used to enable or disable a facility by which a web browser can be used to view the current contents of the target device's display. This facility is very useful when remotely diagnosing problems that an operator may be having with the operator panel or the machine it controls.
- The *Remote Control* property is used to enable or disable an option by which the remote viewing facility is extended to allow a web browser to be used to simulate the pressing of keys or display primitives, thereby allowing remote control of the panel or the machine it controls. While this feature is extremely useful, care must be taken to use the various security parameters to avoid unauthorized tampering with a machine. The use of an external firewall is also strongly recommended if the panel is reachable from the Internet.
- The *Custom Site* property is used to enable or disable a facility by which files stored in the `\WEB` directory of the CompactFlash card are exposed via the web server. This facility is described in more detail below.
- The *Remote Refresh* property represents the frequency at which the web browser connected to the web server will refresh the remote view page. A value of zero will result in refreshes being performed as quickly as possible. Higher values will reduce bandwidth usage, and may be suitable for modem connections.

SECURITY PROPERTIES

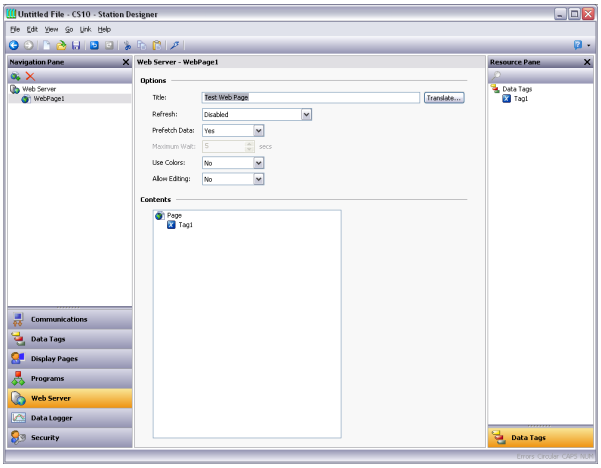


- The *IP Restrictions* group is used to restrict web server access to hosts whose IP address matches the mask and data indicated. All access may be restricted, or the filter may be used to restrict just the remote control or data editing facilities.
- The *Authentication* group defines whether users connecting to the web server will be required to provide username and password information, and how that information will be validated. The *Method* defines the algorithm to be used, with Digest being the recommended choice. The *Source* property is used to indicate whether you will enter the required username and

password directly into the web server properties, or whether you will create users within Crimson's security system and grant them access to the web server.

ADDING WEB PAGES

In addition to the facilities described above, the web server supports the display of generic web pages, each of which contains a predefined list of tag values. These pages are created in the Navigation Pane in the usual way. Each web page has the following properties…



- The *Title* property is used to identify the web page in the menu presented to the user via their web browser. Although the title is translatable, current versions of Crimson use only the US version of the text.
- The *Refresh* property is used to indicate whether or not the web browser should be instructed to refresh the page contents automatically. Update rates between 1 and 8 seconds are supported. Note that the amount of flicker exhibited by the web browser will vary according to the exact package used and the performance of the machine being employed. The update is not intended to be flicker-free.
- The *Use Colors* property is used to indicate if colors defined by a tag's coloring should be used when rendering this page. If enabled, the color shown in the web browser will change depending on the tag status. Refer to the Using Data Tags chapter for more details.
- The *Allow Editing* property is used to enable the editing of data tags via this page. If it is enabled, each data value will have an Edit button displayed, allowing the user to change that value. If the tag has security settings defined, the user logged on to the web server must have sufficient rights to modify the tag. The use of authentication is recommended when using this feature.
- The *Contents* property is used to indicate which tags should be included on the page. Tags can be dragged into the list from the Resource Pane, and moved up and down within the list using standard drag-and-drop techniques.

USING A CUSTOM WEB SITE

While the standard web pages provide quick-and-easy access to the data within the terminal, you may find that your inability to edit their precise formatting leaves your artistic capabilities somewhat frustrated. You may thus use Crimson's custom site facility to create a completely custom web site using your favorite third-party HTML editor, and, by inserting certain special sequences and storing the resulting files on the device's CompactFlash card, publish this site using the target device's web server.

CREATING THE SITE

The web site may use any HTML facilities supported by your browser, but must not use ASP, CGI or other server-side tricks. The filenames used for the HTML files and associated graphics must also comply with the 8.3 naming convention. This means that file extensions will be, for example, `HTM` instead of `HTML` and `JPG` instead of `JPEG`. This also means that the body of the filename must be eight characters or less, and that you must not rely on the difference between upper- and lower-case to differentiate between pages. You may use any directory structure, as long as you once again ensure that your directories observe the 8.3 naming convention and do not rely on case differences.

EMBEDDING DATA

To embed tag data within a web page, insert the sequence `[[N]]`, replacing `N` with the index number or name of the tag in question. When the web page containing this sequence is served, the sequence will be replaced by the current value of the tag, formatted according to the tag's properties. Each tag's index number is displayed on the status bar when a tag is selected within the Data Tag category, and more-or-less corresponds to the order in which the tags were created. Index numbers provide faster access to tag data than names, but do make the web page harder to design. To select an element from an array tag, follow the index number or name with a colon and the array element number. For example, the sequence `[[Tag1:10]]` will display the eleventh (think about it!) element in the array called Tag1.

DEPLOYING THE SITE

To deploy your custom web site, copy it into the `\WEB` directory on the CompactFlash card to be installed in the target device. To copy the files, either mount the card as a drive as described in the early chapters of this manual, or use a suitable card writer connected to your PC. Enable the custom site in the web server's properties, and the site will appear on the main web menu. When the site is selected, a file called `DEFAULT.HTM` within the `\WEB` directory will be displayed. Beyond that point, navigation is according to the links within the site.

USING THE SECURITY SYSTEM

Crimson contains powerful features that allow you to define which operators have access to which display pages, and limit those operators able to make changes to specific data. The software also contains a security logging facility that can be used to record changes to data values indicating when the change occurred and by whom it was performed.

SECURITY BASICS

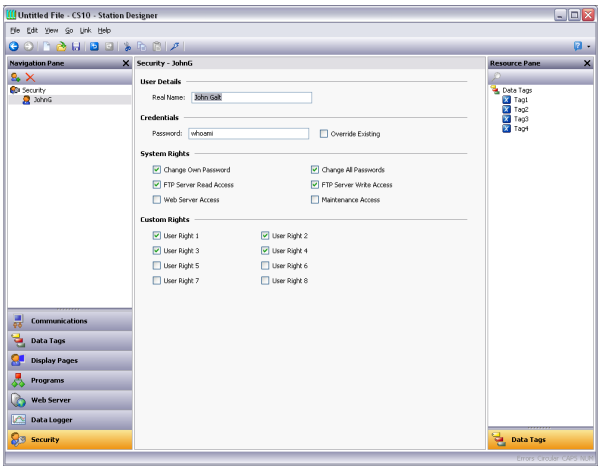
The follow sections detail some of the basic concepts used by the security system.

OBJECT-BASED SECURITY

Crimson's security system is object-based. This means that security characteristics are applied to a display page or to a tag, and not to the user interface element that accesses the page or makes a change to the tag. The alternative subject-based approach typically means that you have to be careful to apply security settings to every single user interface element that might change restricted data. Crimson's approach avoids this duplication and ensures that once you have decided to protect a tag, it will remain protected throughout your database.

NAMED USERS

Crimson supports the ability to create any number of users, each of whom will have a username, a real name and a password. The username is a case-insensitive string with no embedded spaces that is used to identify the user when logging on, while the real name is typically a longer string that is used within log files to record the human-readable identity of the user making a change. Note that you are free to use these fields in other ways if it suits your application: You may, for example, create users that represent groups of individuals or perhaps roles, such as Operators, Supervisors and Managers. You may also decide to use the real name to hold an item such as a clock number to tie user identities into your MRP system.



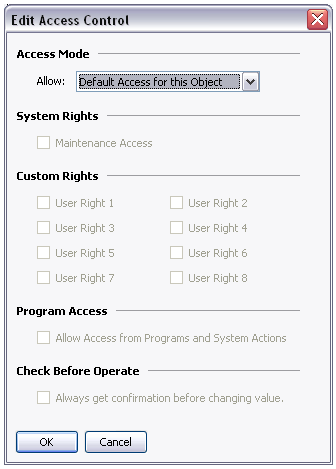
USER RIGHTS

Each user is granted zero or more access rights. A user with no rights can access those objects that merely require the identity of the user to be recorded, whereas users with more rights can access those objects that demand those rights to be present. Rights are divided into System Rights and User Rights, with the former controlling access to facilities within the Crimson software, and the latter being available for general use. For example, User Right 1 might be used within your database to control access to production targets. Only users whom you want to be able to vary such things would then be assigned this right.

ACCESS CONTROL

Objects that are subject to security have an associated *Access Control* property.

Editing the property displays the following…

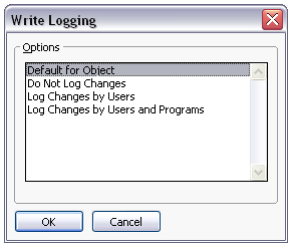


These settings allow you to specify whether the item can be accessed by anyone, by any operator whose identity is known, or by users with specific user rights. You may also specify whether a tag can be changed by a program that is running as a result of something other than user action. This facility allows you to guarantee that no background changes occur to sensitive data, even if a programming error attempts to make such a change.

WRITE LOGGING

Tags also have a *Write Logging* property.

Editing the property displays the following…



The selection indicates whether changes made to a tag by users or by programs should be logged. This facility allows you to create an audit trail of changes to your system, thereby

simplifying faultfinding and providing quality-control information as to process configuration. Note that care should be taken when logging changes made by programs, as certain database may log unmanageable amounts of data in such circumstances.

DEFAULT ACCESS

To speed the configuration process, Crimson also provides the ability to specify default access and write logging parameters for mapped tags, internal tags and display pages. The differentiation between mapped and unmapped tags is important in systems where all changes to external data must be recorded, but where data internal to Crimson can be manipulated without the need for such an audit trail.

ON-DEMAND LOGON

Crimson's security system supports both conventional and on-demand logon. A conventional logon can occur when a user interface element such as a pushbutton is used to activate the Log On User action or to call the `UserLogOn()` function. On-demand logon occurs if the operator attempts an action without sufficient access rights, and if a failed logon attempt has not occurred within the same action. For example, a user may press a button that runs a program to reset a number of values. As soon as the program attempts to change a value that requires security access, the system will prompt for logon credentials. This method reduces operator interaction and produces a more responsive system.

MAINTENANCE ACCESS

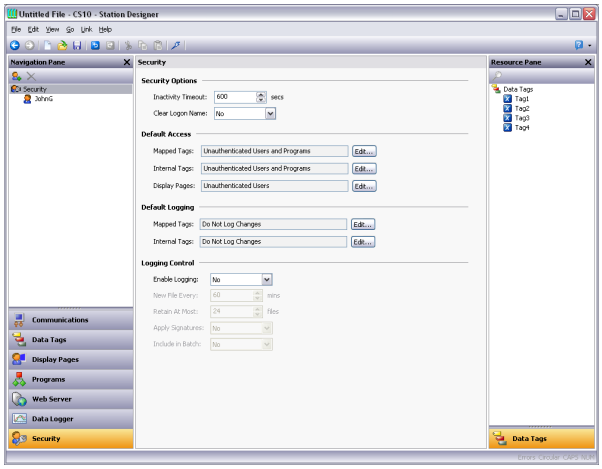
The system also provides a facility called Maintenance Mode to allow the user inactivity timeout to be overridden during system commissioning. This mode is activated if a display page is marked as being accessible with the Maintenance Access right, *and if the current user has gained access to the page as a result of that right.* Use of this mode avoids the need to logon repeatedly when testing the system.

CHECK BEFORE OPERATE

The Check Before Operate feature allows you to force the user to confirm every change to a particular sensitive data item. The feature enabled by selecting the appropriate setting on a data tag's security descriptor. When a change is made to a tag that has this feature enabled, a popup will appear displaying the old and new values and demanding confirmation before the change is permitted. This feature operates whether or not a user is currently logged on, and is in addition to any user rights required for the change to occur. It is also independent of the action Protection operations defined when creating the user interface.

SECURITY SETTINGS

The security system settings are accessed via the root item in the Security category…

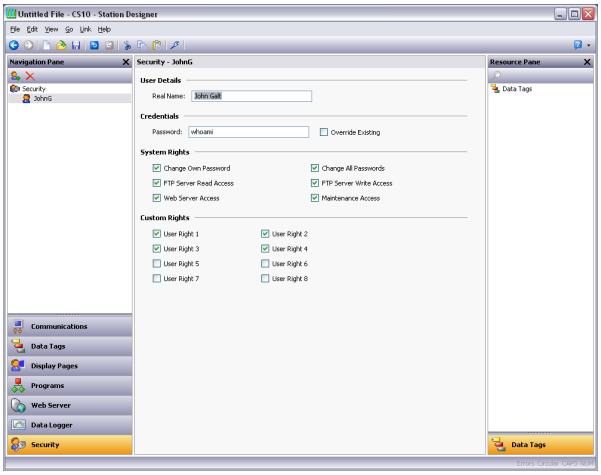


The available properties are as follows…

- The *Inactivity Timeout* property is used to indicate how much time must pass without user input before the current user is automatically logged off. Too high a value for this setting will produce an insecure system, while too low a value will produce a system that is awkward for operators.
- The *Clear Logon Name* property is used to indicate whether or not the username should be cleared before asking the operator to logon. If this setting is disabled, the previous username will be displayed, and only the password will need to be re-entered. Enabling this feature produces higher security, and may be required to comply with security standards in certain industries.
- The *Default Access* properties are used to indicate the access to be provided to various objects should no specific access be defined for that item. The settings are as described in the Access Control section above.
- The *Default Logging* properties are used to indicate whether changes to mapped and unmapped tags should be logged should no specific logging criteria be defined for a tag. It is not possible to log programmatic access by default, as such logging should be carefully considered to avoid excessive log activity.
- The *Logging Control* properties define whether and how the security logs should be created. Refer to the Using the Data Logger chapter for information on how the data is written and how files are named.

CREATING USERS

Users are created and otherwise manipulated via the usual methods in the Navigation List…



Each user has the following properties…

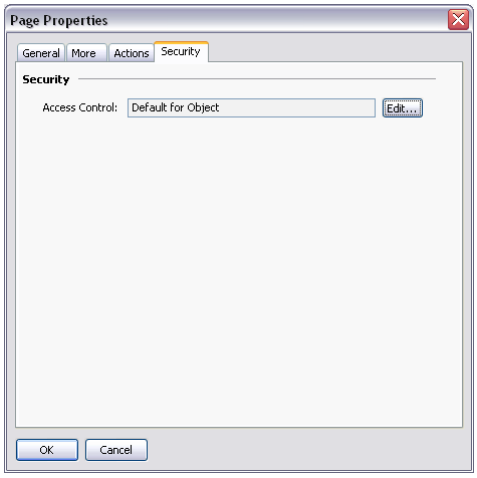
- The *Real Name* property is used to record the user's identity in security logs, and is also shown in the Security Manager primitive that is used to change passwords at runtime. If maximum security is required, the user name should not be easily derived from the real name.
- The *Password* property specifies an initial password for this user. The password is case-sensitive and comprises alphanumeric characters. Note that if the *Override Existing* box is checked, any changes made to this password from the target device will be overridden when this database is downloaded.
- The *System Rights* properties are used to grant a user the ability to perform certain system actions. The properties relating to password changes are self-explanatory, while the user of Maintenance Mode is described above.
- The *Custom Rights* properties are used to grant a user certain rights which may then be used within the database to allow access to groups of tags or display pages. The exact usage of these rights is up to the system designer.

SPECIFYING TAG SECURITY

Each tag has a tab called Security which defines the access control and write logging settings for that tag. If you do not define specific settings, the system will use the appropriate default settings, depending on whether it is mapped to external data.

SPECIFYING PAGE SECURITY

The access control settings for display pages are defined via their Properties dialog…



Once again, if no setting is defined, default settings will be used.

SECURITY RELATED FUNCTIONS

Please refer to the Reference Manual for details on the `UserLogOn()`, `UserLogOff()` and `TestAccess()` functions. This last function is useful when changing many values from within a program, as it allows you to force an access check early in the code to avoid making changes only to have later operations fail due to insufficient user rights.

USING SERVICES

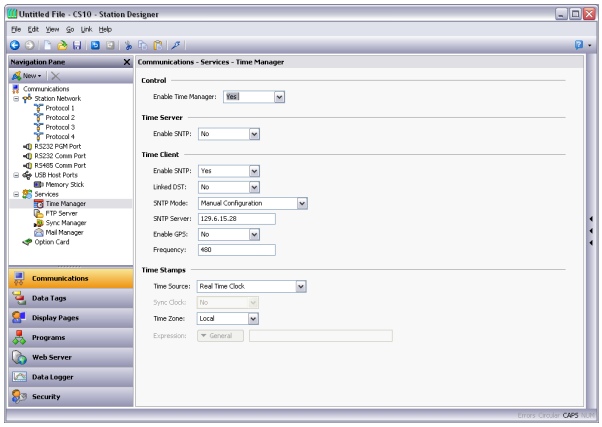
In addition to the core functions described earlier in this document, the Communications category also allows various services to be configured. These services appear in the Navigation Pane under the Services icon, and each is described below.

USING TIME MANAGEMENT

Crimson contains facilities to allow you to synchronize the time and date within the target device with a variety of sources. The Time Manager can also maintain information about the device's time-zone, and whether daylight saving time is currently enabled. In fact, having accurate time-zone information available is vital to proper synchronization, as the various synchronization methods are all designed to work with Universal Coordinated Time, also known as UTC or Greenwich Mean Time. Crimson can act both as a client and a server, either requesting the time or providing the time to other Crimson-based devices. Note that the server implementation does not currently support third party clients.

CONFIGURING THE SERVICE

The Time Manager is configured via the associated icon in the Navigation Pane…



The *Enable Time Manager* property is used to control access to the other facilities. If it is not checked, Crimson will operate in the local time zone only and will have no knowledge of time-zones or other time management information.

TIME SERVER

Appropriately configuring the Enable SNTP property of the Time Server section will instruct Crimson to act as an SNTP server. This will allow other Crimson devices to synchronize their own clocks to the clock of this unit. Note that Crimson's implementation of SNTP is not fully RFC compliant, and is not supported as a source of synchronization for third-party clients.

TIME CLIENT

Selecting Yes in the Enable SNTP property of the Time Client section will instruct Crimson to run its SNTP client. Crimson will then attempt to synchronize its clock with another

Crimson-based device, or to another network-accessible SNTP time source such as a computer running Windows XP. The time client has the following additional properties…

- The *Linked DST* property is used to instruct the SNTP client to attempt to read the current Daylight Savings Time setting from the SNTP server. As this facility is not a standard part of the SNTP protocol, it will only operate if another Crimson device is specified as the server. The facility is useful, in that it allows the Daylight Savings Time adjustment to be made via a single device on the factory network, with the other devices then following the central setting.
- The *SNTP Mode* and *SNTP Server* properties are used to configure the IP address of the Simple Network Time Service server. If Configured via DHCP is selected, at least one Ethernet port must be configured to use DHCP, and the server must be configured to designate a server via option 42.
- The *Enable GPS* property is used to instruct the time client to use a GPS unit connected via NMEA-0183 as an alternative method of obtaining the current time. The unit may be connected to any serial port using the appropriate driver.
- The *Frequency* property specifies how often Crimson should attempt to synchronize its time by the methods enabled above. Crimson will always attempt to sync twenty seconds after power-up, and will then sync as specified by this property. If a given attempt to sync fails, the unit will retry every 30 seconds until it is successful in finding a suitable time source.

TIME STAMPS

Crimson can record a variety of log files on the target device's CompactFlash card, and each log entry has a time stamp. By default, the time stamp comes from the local real time clock and is in the local time zone. The behavior can be changed via the following properties…

- The *Time Source* property is used to indicate from where the time stamps should be obtained. The default setting obtains the time from the unit's own real time clock, while the alternative allows the use of an expression to define the current time. This expression is typically a reference to a data item in a connected device, allowing that device's clock to be used for data logging. The expression must be entered in the *Expression* property.
- The *Sync Clock* property is used to indicate whether the local real time clock should be synced to the alternative time source specified above. If this option is enabled, the local clock will be synchronized on startup and periodically thereafter, and will be used as a time stamp source if the alternative source is not available due to comms problems.
- The *Time Zone* property is used to indicate the time zone to be used for time stamps. It is only applicable when the local real time clock is configured as the source for time stamps. Selecting Local will use the local time zone; selecting UTC will use Universal Coordinated Time instead. This latter setting produces log files which are more easily portable across time-zones, and which do not suffer from discontinuities when switching in and out of Daylight Savings Time.

CHOOSING AN SNTP SERVER

When configuring the SNTP client, you have several options when selecting a server.

If you have a Windows- or Unix-based time server as part of your network infrastructure, you should ultimately synchronize to this source to ensure enterprise-wide synchronization. If you have several Crimson devices on the same network, though, you will find it better to nominate one of these as the master device for the purpose of setting Daylight Savings Time, and then have that device alone synchronize to the enterprise time source. You can then configure the other devices to synchronize to the master device, and enable the Linked DST facility to propagate the Daylight Savings Time setting around your factory.

If you have no enterprise time source available, you may choose to nominate a single Crimson device as the point where an operator will set the time, and then have other devices synchronize to that source. Alternatively, if your installation provides TCP/IP access to the Internet via either Ethernet or a modem connection, you may configure the SNTP client to synchronize to a public time server. An example of this would be 192.6.15.28, which is the current IP address of a public time server provided by NIST.

A list of other servers can be found at…

http://support.microsoft.com/kb/262680

Note that since Crimson uses an IP address and not a host name to reference the SNTP server, it will lose connection with any server that is relocated to a new network address. While such relocations are very rare, they are beyond your control and that of Red Lion. The use of an enterprise time source which accesses its own source via DNS is thus considered preferable!

TIME-ZONE CONFIGURATION

As mentioned above, a Crimson device must have knowledge of the current time-zone if it is to use advanced time management. This information can be provided in two ways. The easiest method is to use the Send Time command on the Link menu of the Crimson configuration software. In addition to setting the clock, this command also sends the PC's current time zone and the status of Daylight Savings Time. Crimson will store this data in non-volatile memory, and use it from that point forward. Obviously, you should be sure that the PC contains valid time and date information before sending it to the unit!

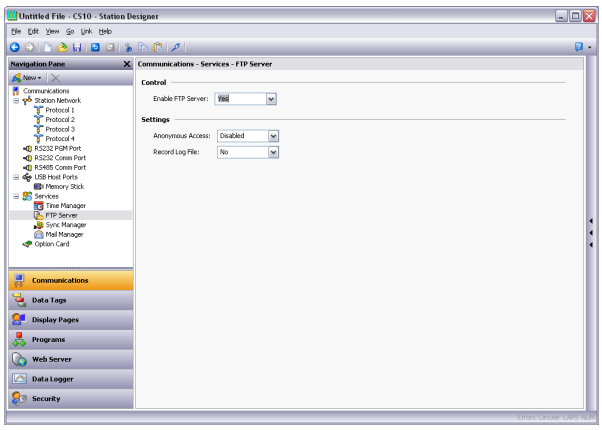
The alternative method is to use the system variables `TimeZone` and `UseDST`. The former holds the number of hours by which the local time zone differs from UTC, and may be either negative or positive. For example, a setting of –5 corresponds to Eastern Standard Time in the United States. The latter contains either 0 or 1, depending on whether Daylight Savings Time is active. Editing either of these variables via the user interface will result in the unit's clock changing to take account of the new settings. For example, enabling Daylight Savings Time will move the clock forward one hour, while disabling it will move it back. A typical database will only need to expose `UseDST` for editing by the user, and even this may not be necessary if the Linked DST facility described above is in use.

USING THE FTP SERVER

Crimson's FTP server provides a method to exchange files between a Crimson device and a remote computer running an FTP client application. The Crimson device will act as a server, waiting for client applications to connect and download or upload files.

CONFIGURING THE SERVICE

The FTP Server is configured via the associated icon in the Navigation Pane…



The following properties can be configured…

- The *Anonymous Access* property defines the rights—if any—granted to a user accessing the server using anonymous FTP. A setting of Disabled will prevent anonymous access. A setting of Read-Only will allow the user to download files from the CompactFlash card, but will prevent uploads. A setting of Read-Write will allow both uploads and downloads.
- Enable the *Record Log File* to keep a log of all FTP interactions in the root directory of the CompactFlash card. This file can be useful when debugging FTP operations, but it will tend to degrade performance slightly.

FTP SECURITY

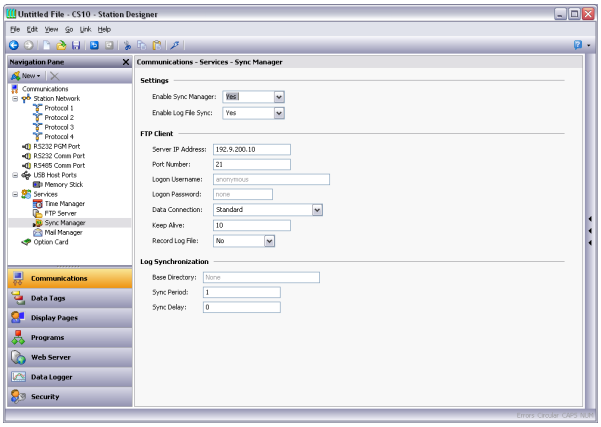
As the FTP Server can provide full access to the CompactFlash card, it is high recommended that you use the Security Manager to define specific username and password combinations and to grant those users the appropriate access rights. In general, you should avoid granting anonymous access, and you should especially avoid allowing anonymous writes.

USING FILE SYNCHRONIZATION

The Synchronization Manager can be used to exchange files between a Crimson device and an FTP server. This facility can be used to synchronize log files with a server computer, either automatically or on-demand, thereby providing an alternative to accessing the log file via the web server, and allowing for unattended transfer of files from many stations to a central point. (Note that although it is called the Synchronization Manager for historical reasons, this service is actually based upon a general-purpose FTP client that can also be used to perform other FTP transfers, whether or not log files are being synchronized.)

CONFIGURING THE SERVICE

The Synchronization Manager is configured via the associated icon in the Navigation Pane…



FTP CLIENT

The following properties relate to the FTP client…

- The *Enable Sync Manager* property is used to enable the FTP client. The client may be enabled without actually enabling synchronization, allowing it to be used for manual file transfer via the `FtpFilePut()` and `FtpFileGet()` functions.
- The *Enable Log File Sync* property is used to enable actual synchronization. See the next section for details of the other settings related to this feature.
- The *Server IP address* property is used to indicate the IP address of the server.
- The *Port Number* property is used to indicate the TCP port to which the FTP client service will attempt to connect. The default value is suitable for most applications, as most servers will listen on port 21.
- The *Logon Username* and *Logon Password* are the credentials that are submitted to the server when the connection is established. Both are typically case sensitive, although that depends on the server implementation. For anonymous login, leave the Username at its default value, and either leave the password blank, or enter your email address as a courtesy to the server provided.

- The *Data Connection* provides a choice between standard and PASV mode. You can enable the PASV mode to have the FTP client initiate all data connections rather than waiting for incoming connections from the server. This mode is sometimes required when working behind non-FTP aware firewalls or when operating via certain forms of network address translation. Typically, it is also used when working over a GPRS modem connection.
- The *Keep Alive* time is the period for which the FTP connection should be kept alive in case further transfers are required. A value of zero will close the connection as soon as the current transfer has been completed. Non-zero values make for more efficient operation when transferring multiple files.
- The *Record Log File* property can be used to keep a log of all FTP interactions in the root directory of the CompactFlash card. This file can be useful when debugging FTP operations, but it will tend to degrade performance slightly.

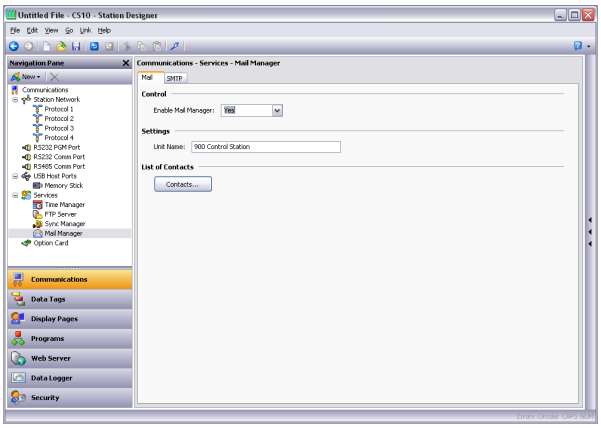
LOG SYNCHRONIZATION

The following properties relate specifically to log file synchronization…

- The *Base Directory* property defines the directory on the server where the log files will be placed. This directory is relative to the FTP server's folder space, not to the underlying directory structure of the server's own filing system. You will typically specify a different base directory for each Crimson device that is synchronizing to a given server.
- The *Sync Period* property specifies how often the FTP client will connect to the server and transfer its files. It is measured in hours, and is always based from midnight, such that selecting a value of three will result in transfers at midnight, 3:00 am, 6:00 am and so on.
- The *Sync Delay* property defines an offset in minutes from the standard time at which file transfers will occur. This property can be used to allow multiple terminals to talk to one server without all the file transfers occurring at once, and thereby overloading the target's capabilities.

USING ELECTRONIC MAIL

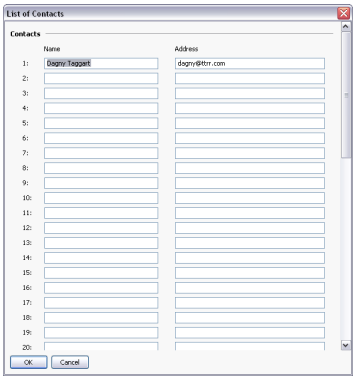
Crimson can be configured to send email messages when alarm conditions are present, or when notifications need to be provided of other events within the system. The mail transports and the email address book are configured via the Mail Manager…



The properties on the Mail tab are used to enable or disable the mail manager, and to provide a name for the device on which Crimson is running. This name will be used within email messages to identify the originator of the message. Applications will typically use the name of the machine to which the device is attached, or the name of the site that it is monitoring.

ADDING CONTACTS

The Contacts button can be used to access Crimson's address book…

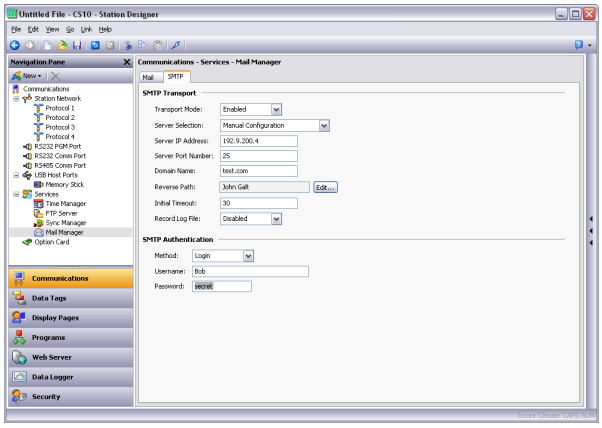


Each entry allows a Display Name and an Address to be entered. The address should be in a format suitable for the required transport. For example, SMTP names should be in the usual `name@domain` format, while SMS names should be entered as international-format telephone numbers without the leading plus sign. Multiple email addresses can be entered by separating them by semicolons, allowing simple mailing lists to be created.

CONFIGURING SMTP

The SMTP tab is used to configure the Simple Mail Transport Protocol. This is the standard protocol used to send email over the Internet or over other TCP/IP networks. SMTP addresses follow the familiar `name@domain` standard.

The configuration options for the SMTP transport are shown below…



SMTP TRANSPORT

- The *Transport Mode* property is used to enable or disable the transport. Note that the mail manager must be enabled via the Mail tab before the SMTP transport can be enabled. Note also that at least one transport must be enabled if the mail manager is to be able to deliver messages.
- The *Server Selection* property defines how the transport will locate an SMTP server. If Manual Selection is used, the *Server IP Address* property should be used to manual designate a server. If Configured via DHCP is selected, at least one Ethernet port must be configured to use DHCP, and the server must be configured to designate an SMTP server via option 69.
- The *Server IP Address* property is used to designate an SMTP server when manual server selection is enabled. The server must be configured to accept mail from the panel, and to relay messages if required by the application.
- The *Server Port Number* property defines the TCP port number that will be used for SMTP sessions. The default value is 25. This value will be suitable for most applications, and will only need to be adjusted if the SMTP server has been reconfigured to use another port.
- The *Domain Name* property specifies the domain name that will be passed to the SMTP server in the HELO or EHLO command. The vast majority of SMTP servers ignore this string. In the unlikely event that your SMTP server attempts to do a DNS lookup to confirm the identity of its client, you may need to enter something appropriate to your DNS configuration.
- The *Reverse Path* property specifies the email address that will be supplied as the originator of the messages sent by the target device. The property comprises a display name, and an email address. Since Crimson is not capable of receiving messages, the email address will often be set to something that will return an "undeliverable" message if a reply is sent.
- The *Initial Timeout* property specifies how many seconds the mail client will wait for the SMTP server to send its welcome banner. Some Microsoft servers attempt to negotiate Microsoft-specific authentication with mail clients, thereby delaying the point at which the

banner appears. You may want to extend this time period to 2 minutes or more when working with such servers.

- The *Record Log File* property can be enabled to keep a log of all SMTP interactions in the root directory of the CompactFlash card. This file can be useful when debugging SMTP operations, but enabling it all the time will tend to degrade performance slightly. A log file can only be created when a connection with the time server has been established.

SMTP AUTHENTICATION

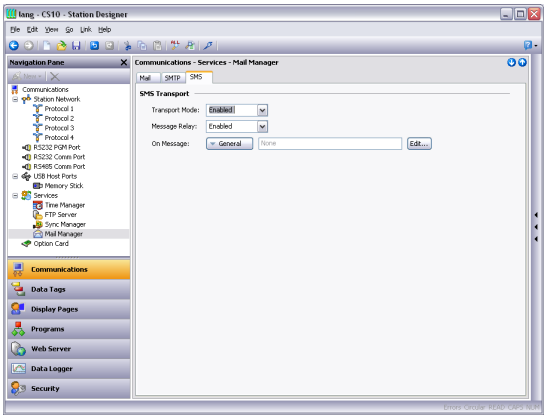
- The *Method* property indicates the type of authentication to be attempted by the client. A selection of Digest will insist upon a authentication technique that sends the password in an encrypted form, and will skip authentication if the server does not support such a method. A selection of Basic will attempt to use the secure technique, but will fall back to trivially encoded transmission if necessary. A selection of None will not attempt to authenticate. Your server may or not require authentication. Contact you network administrator or mail provider for more details of the setting that should be used for your server.

- The *Username* and *Password* properties provide the optional credentials for the authentication process described above.

CONFIGURING SMS

The SMS tab is used to configure the Short Message Service transport that is supported when using a GPRS modem in association with the target device. Email addresses for the SMS transport are in the form of international-format telephone numbers without the plus sign. For example, an address of 17175551111 would send a message to the cell phone or other GSM device that had a number of (717) 555-1111 within the United States.

The configuration options for the SMS transport are shown below…



- The *Transport Mode* property is used to enable or disable the transport.

- The *Message Relay* property is used to enable or disable Crimson's SMS relay feature. If this feature is enabled, a user who receives an SMS message that has been sent to several recipients can reply to that message, and have the Crimson device runtime relay the message to the other recipients. This provides a simple conferencing facility between message recipients.

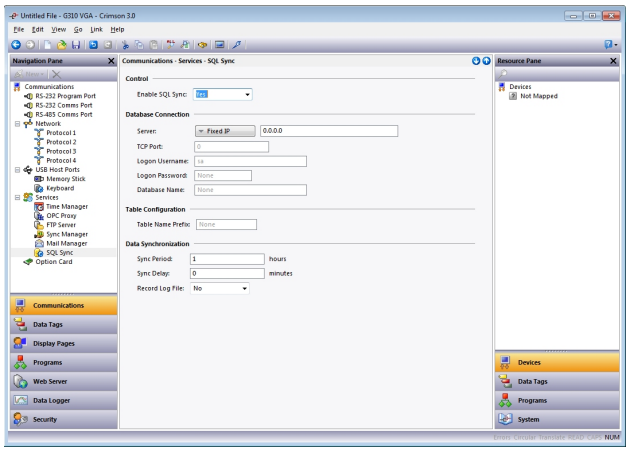
- The *On Message* property is used to define an action to be executed each time a message is received. A local system variable called `Data` is defined within the action, allowing access to the message itself. The source number of SMS is prefixed to the message, with a colon separating it from the message itself.

USING SQL SYNC

SQL Sync can be used to set up a connection to an SQL server to periodically synchronize all log files to a remote SQL database.

CONFIGURING THE SERVICE

SQL Sync is configured via the associated icon in the Navigation Pane…



DATABASE CONNECTION

The following properties relate to the database connection…

- The *Server* property is used to specify the IP address of the SQL server.
- The *TCP Port* property is used to indicate the TCP port to which the SQL Sync will attempt to connect. This port must agree with the port configured for TCP/IP access on the SQL Server.
- The *Logon Username* and *Logon Password* are the credentials that are submitted to the server when the connection is established. Both are typically case sensitive, although that depends on the server implementation.
- The *Database Name* property is used to specify the name of the database on the server to which SQL Sync will synchronize.

TABLE CONFIGURATION

The following property relates to table configuration…

- The *Table Name Prefix* property is used to specify a namespace for all tables created by SQL Sync. This prefix will be prepended to the name of the log using an underscore to create the table name.

DATA SYNCHRONIZATION

The following properties relate to data synchronization…

- The *Sync Period* property is used to specify how frequently SQL Sync will synchronize with the server. The starting period begins at midnight, so, for example, a Sync Period of 3 will synchronize first at 12:00, then again at 3:00.
- The *Sync Delay* property is used to specify how many minutes after the time determined by the *Sync Period* that SQL Sync should wait before synchronizing. For example, with a *Sync Period* of 2 and a *Sync Delay* of 5, SQL Sync will synchronize every 2 hours at 5 minutes after the hour.
- The *Record Log File* property is a flag to determine whether or not SQL Sync activity should be logged to the compact flash. Information logged includes commands sent to the SQL server and messages about any errors that occur during synchronization. This feature can be useful for debugging failed synchronization attempts.

SHARING PORTS

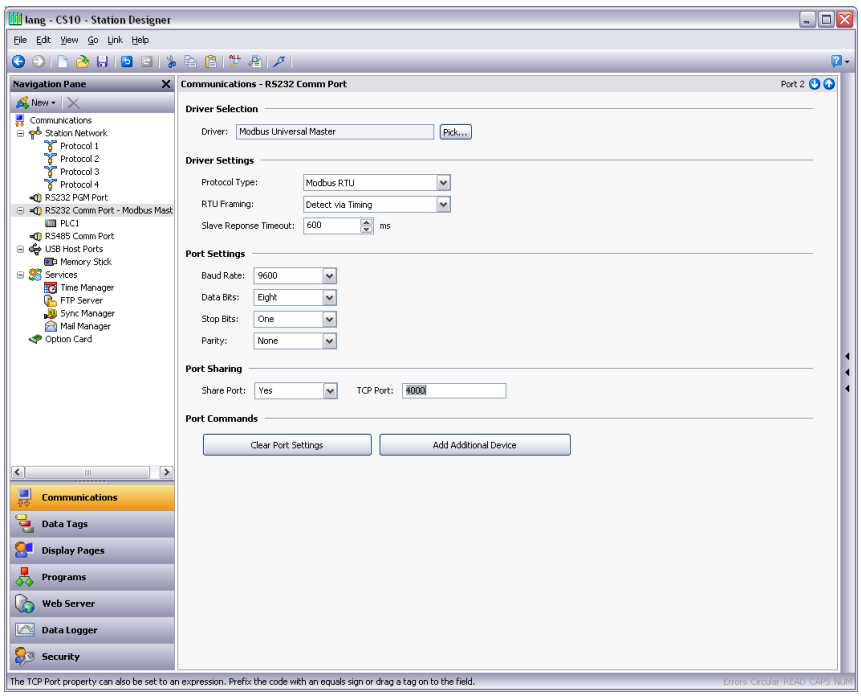
Crimson provides a port sharing facility that allows either physical or virtual serial connections to be made to any serially-connected device. For example, you may be using an operator panel with a programmable controller, but since the PLC has only a single serial port, you may find yourself swapping cables when modifying the ladder program. By sharing the communications port to connect to the PLC, you can send data directly to the controller, either from another serial port or by means of a connection made over a TCP/IP link.

ENABLING TCP/IP

The first configuration step when using port sharing is to enable the Ethernet port as described elsewhere in this manual. While you may not choose to use the virtual serial port facility, even the local sharing of ports is based upon the TCP/IP protocol, which will not be available unless at least one network interface is enabled.

SHARING THE REQUIRED PORT

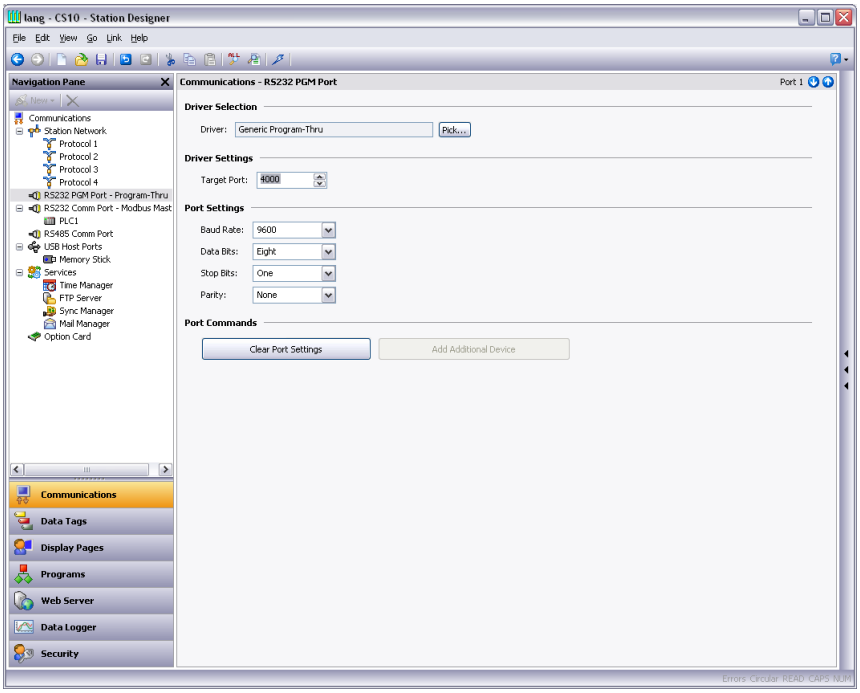
The next step is to share the required port, which is done by selecting Yes in the Share Port property and then entering a suitable TCP/IP port number to indicate exactly how the virtual port should be addressed…



If you leave the port setting at zero, a number of 4000 plus the logical index of the port will be used. You may use any number that is not already used by another TCP/IP protocol. If you are stuck for ideas, we recommend numbers between 4000 and 4099.

CONNECTING VIA ANOTHER PORT

If you want to use another port on the target device to route data to the shared port, you must select the Generic Program Thru driver for that port, and configure this driver with the TCP/IP port number of the port that you have shared. In the example below, we are routing data from the programming port to a PLC that is connected via the RS-232 comms port…



Note that in most cases, the Baud rate and other port settings do not have to be the same as those for the port that we are sharing, as Crimson will perform the conversion. The one exception to this is where one device transmits large bursts of data without any replies from the other. In this case, the device carrying out the larger transmissions must not be using a higher Baud rate than the device receiving them, or Crimson may not have enough memory to buffer the data while waiting for it to be retransmitted.

In the example above, to make use of the shared port you would connect a spare serial port on your PC to the programming port of the target device, and configure the PLC programming software to talk to this COM port. As soon as the PC begins to send to the PLC, all communications between Crimson and the PLC will be suspended, and the target device's two ports will be connected in software, such that the PC will appear to be talking directly to the PLC. If no data is transferred for more than a minute, communications between Crimson and the PLC will be resumed.

CONNECTING VIA ETHERNET

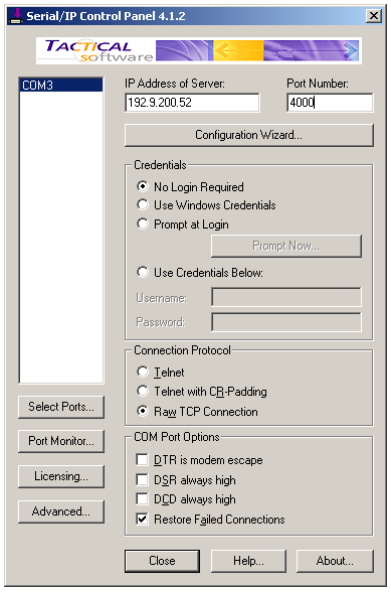
Rather than using an additional serial port on your PC and on the Crimson-based device, it is possible to use a third-party utility to create what are known as virtual serial ports on your PC. These appear to applications to be physical COM ports, but they in fact send and receive data to a remote device over TCP/IP. By installing one of these utilities and configuring it to address the Crimson-based device, you can have serial access to any devices connected to that

device without any additional cabling. Indeed, there is no need to have any physical serial ports available on the PC at all—something that is very valuable when working with modern laptops, where a COM port is often an expensive option.

Several third-party virtual serial port utilities are available. On the freeware side, a company known as HW Group (http://www.hw-group.com) provides a utility called HW Virtual Serial Port. There are also a number of other freeware port drivers available, most of which seem to be derived from the same source base. On the commercial side, a company called Tactical Software (http://www.tacticalsoftware.com) offers Serial/IP for about $100 a port.

While the various freeware drivers no doubt have many contented users, we have found that these drivers have occasional stability problems on certain PCs. Tactical Software's Serial/IP is thus the only package that we are able to support, and the following information assumes that you are using this package.

To create a virtual serial port, open Serial/IP's configuration screen, and select the name of the COM port you wish to define. This will typically be the first free COM port after those allocated to the physical ports and modems installed in your PC. Next, enter the IP address of the Crimson-based device, and enter the TCP/IP port number that you allocated when sharing the port. The example below is configured as required by the previous samples in this document. Finally, ensure Raw TCP Connection is selected, and close the Serial/IP dialog.



You will now be able to configure any Windows-based software to use the newly-created COM port for download. When the software opens the connection, Crimson will suspend communications on the shared port, and data will be exchanged between the PC software and the remote PLC—just as if they were connected directly! When the port is closed, or if no data is transferred for a minute, communications will be resumed.

Note that assuming you have purchased the appropriate number of licenses for Serial/IP, you will be able to create as many virtual ports as you need. This means that you can be connected to multiple devices from the same PC, downloading to each via its respective programming

package—all without plugging or unplugging a single cable. This feature is extremely valuable when you have many devices in a complex system.

PURE VIRTUAL PORTS

In some circumstances, you may want to use a spare serial port on a Crimson-based device to provide access to a remote device that is not otherwise referenced in your database, effectively using the spare port as a remote serial server. To do this, configure the port in the usual way, selecting the Virtual Serial Port driver for that port. Then, share the port as described above, exposing it via TCP/IP. The Virtual Serial Port driver performs no comms activity of its own, but still allows the device to be shared for remote access.

LIMITATIONS

Note that some PLC programming packages may not work with virtually or physically shared ports. Issues to watch out for are tight timeouts that do not allow Crimson time to relay the data to the PLC; a reliance on sending break signals or on the manipulation of hardware handshaking lines; or DOS-style port access such that the package cannot see the virtual serial port. Luckily, these issues are rare, and most packages will happily communicate as if they were directly connected to the PLC in question.

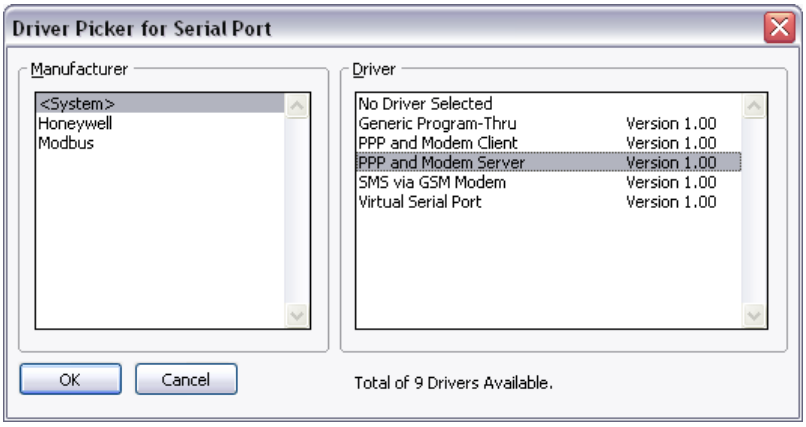
USING MODEMS

This chapter explains how to configure Crimson to work either with modems, or with direct serial connections to computers running the Windows operating system.

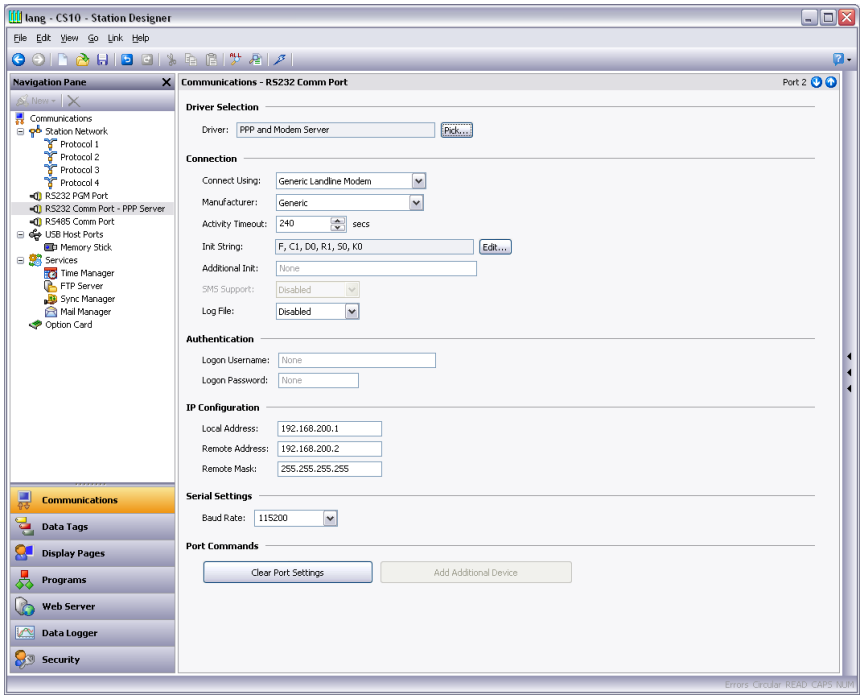
Note that Crimson's modem support is entirely based upon the Point-To-Point Protocol, otherwise known as PPP. While protocols such a Modbus allow a single conversation to occur between any two devices, PPP is more akin to an Ethernet connection in that it allows an unlimited number of logical connections to exist on a single physical link. A single PPP connection can thus allow simultaneous access to the panel's TCP/IP download facility, its web server, its shared serial ports, and to any TCP/IP protocols that have been defined.

ADDING A DIAL-IN CONNECTION

To add a dial-in connection to your database, select the Communications category and navigate to the serial port via which the connection will be made. Click on the Pick button of Driver property, and select the *PPP and Modem Server* driver from the System section…



The Editing Pane will now show the modem configuration…



The modem has the following configuration options…

- The *Connect Using* property is used to select the physical device to be used to make the connection. The devices supported are direct serial connections to computers running the Microsoft Windows operating system, generic landline modems which implement the Hayes command set, and GSM modems implementing the industry standard GSM commands. For dial-in connections, the GSM devices must be configured in Circuit Switched Data mode.

- The *Manufacturer* property is used to select from the manufacturers or models for which specific configurations have been developed and stored within Crimson. Leaving this setting at Generic will allow you to customize the settings related to initialization strings and the like. Please consult Technical Support for the settings required for any particular modem.

- The *Activity Timeout* property is used to define how long a period must pass without the G3 sending a packet over the PPP link in order for the connection to be terminated. For dial-in connections, it is assumed that the connecting device is friendly, so no effort will be made to filter out optional packets that might result in the link staying active for long periods. Note that even if you want a permanent connection, you must enter a suitable timeout so as to allow the detection of dead links. This implies that so-called permanent connections may still drop on occasions, but will in any case be immediately reestablished.

- The *Init String* property is used to enable or disable certain commands during the initialization sequence. It is automatically configured if a specific setting is entered in the Manufacturer property.

- The *Additional Init* string is used with non-direct links, and provides a series of AT commands to be used to initialize the modem. The initial AT prefix is not required. Several

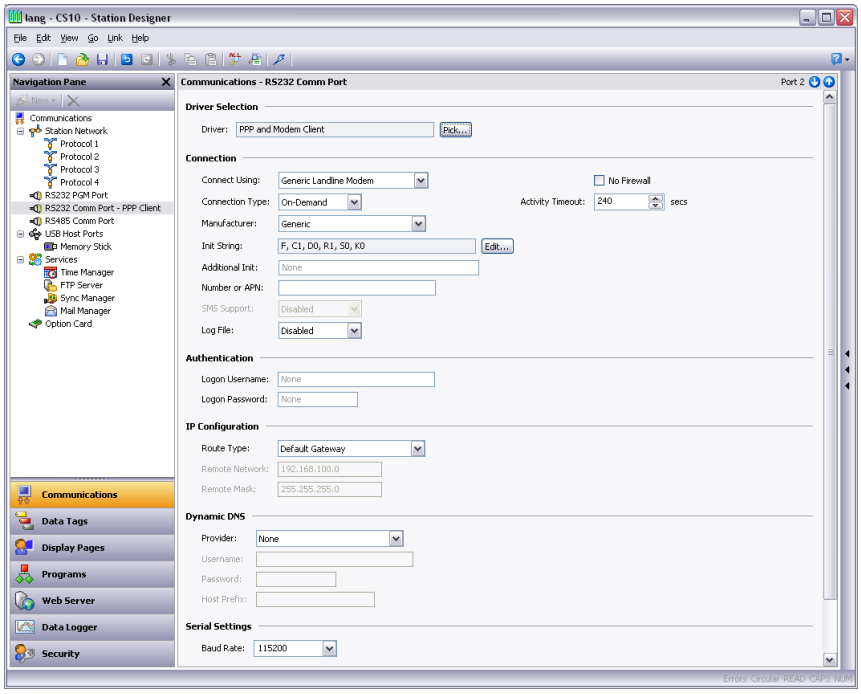
commands may be combined by simply placing one after the other. The exact string that will be required for your modem is dependent upon its internal software, so if you contact Technical Support for assistance, be sure to have exact make and model information available.

- The *SMS Support* property is used to enable Short Message Service messaging when using a GSM modem. In order for SMS messaging to operate properly, you will also have to enable the SMS Transport in the Mail Manager as described elsewhere in this manual.
- The *Logon Username* and *Logon Password* properties are used to define the credentials that the remote client must provide in order to be allowed to connect to this device. The username is not case sensitive, while the password is. Crimson's PPP implementation will ask its peer to use CHAP authentication to avoid transmitting or receiving plaintext password, but will fallback to using PAP if the remote client does not support CHAP.
- The *Local Address* property is used to define the IP address to be allocated to the local end of the connection. This will thus be the IP address of the G3 for this link. Please note that this must not be the same as the IP address of the G3's Ethernet port, as every physical IP interface must have a distinct IP address. The default value will work in most situations, unless your network design demands that you use a different setting.
- The *Remote Address* property is used to define the IP address to be allocated to the remote end of the connection. It is used together with the *Remote Mask* property to determine what packets will be routed to this connection. For most applications, a mask of 255.255.255.255 will be used, thereby instructing Crimson to send via this interface only those packets directly bound for the remote client. A mask of 0.0.0.0, by contrast, will allow all packets that do not specifically match another interface to be forwarded to the remote client, presumably for further forwarding to the intended host. Intermediate masks may be used to control exactly which packets are sent.

ADDING A DIAL-OUT CONNECTION

Dial-out connections are added exactly as described above, except that the *PPP and Modem Client* driver should be selected for the required port.

The configuration options for this modem are shown below…



The modem has the following properties that are distinct from those for dial-in connections…

- The *Connect Using* property is as for dial-in connections, with the addition of support for GPRS connections via a GSM modem. These connections differ from CSD connections in that they achieve much higher speeds, and are typically charged on the basis of how much data is transferred rather than how long the connection is maintained. GPRS connections may thus be configured for permanent connection, unless there is a need to provide downtime to allow SMS messages to be transferred.

- The *No Firewall* property is used to turn off the firewall protection that is otherwise provided for dial-out connections. This protection prevents incoming connections from being made to this interface, and prevents the G3 from sending certain diagnostic packets that might either provide a hacker with information about the system, or might be used by an attacker to keep a connection active in the absence of actual data transfer. If you are connecting directly to the Internet by means of this connection, you should not normally turn off the firewall. The firewall should be disabled only for connections to corporate networks or to other controlled environments.

- The *Connection Type* property is used to indicate whether you want this connection to be permanently maintained, or whether you want it to be established automatically when an attempt is made to transfer data to hosts that are reachable via this interface. If you select an on-demand connection, you must specify the timeout after which the link will be terminated if no packets have been transmitted by the G3.

- The *Logon Username* and *Logon Password* properties are used to define the credentials that will be passed to the remote server when attempting to initialize this connection. The username is not case sensitive, while the password is. Crimson's PPP implementation will ask

its peer to use CHAP authentication to avoid transmitting or receiving plaintext password, but will fallback to using PAP if the remote server does not support CHAP.

- The *Route Type* property is used to define the data that will be transferred via this interface. For on-demand connections, this effectively defines when the connection will be activated. If *Default Gateway* is selected, any packets that do not match the address and network mask of the Ethernet connection will be sent to this interface. Note that in this mode, the Ethernet port must have a gateway setting of 0.0.0.0, or it will take all the packets and leave none to activate the modem! If *Specific Network* is selected, you must provide the address and network mask that defines the network to which packets will be routed.

ADDING AN SMS CONNECTION

SMS connections are used when text messaging functionality is required, but where neither dial-in nor dial-out PPP connections will be established. They are configured as described above, except that the *SMS via GSM Modem* device should be selected for the required port.

The properties for this driver are a subset of those provided for dial-in connections. SMS support is always enabled with this driver, but note once again that for SMS messaging to operate, you will have to enable the SMS Transport within the Mail Manager.

SMS MESSAGE PROCESSING

When SMS messaging is enabled, Crimson will instruct the GSM modem to check for new incoming or outgoing messages every five seconds. Incoming messages are forwarded to the mail manager, which will optionally forward them to other users according to its configuration. Note that it is not possible to check for messages while the modem is connected to a CSD or GPRS session, so you will want to avoid using permanent connections when working with SMS. Note also that if more than one GSM modem is configured, all will be able to receive messages, but only the last modem will be used for sending.

CHECKING THE MODEM STATUS

In order to help debug modem connections, Crimson provides the `GetInterfaceStatus()` function. This function takes a single argument, which is the numeric index of the required interface. Interface zero is the internal loopback interface. Next come any Ethernet interfaces that are enabled, followed by the PPP interfaces. In a system using a single Ethernet port, for example, the first PPP interface will have an index of 1.

The function returns a string, which can be interpreted according to the following table…

STATUS	MEANING
CLOSED	The interface has not yet been initialized. This state will only occur for a short time during system start-up.
INIT	The modem is being initialized. If the connection remains in this state, there are probably errors in the init strings being sent to the modem.

STATUS	MEANING
IDLE	The link is idle. GSM modems will return a number at the end of the string to indicate signal strength. The next table explains how to interpret these values.
SMS	The modem is sending SMS messages, or polling the modem to see if new SMS messages are available. If SMS messaging is enabled for a modem, you will see this state appear for a short period every five seconds.
CONNECTING	The modem is establishing a connection. This state typically appears only for client connections, and indicates that a call is being placed.
LISTENING	The modem is waiting for a call. This state appears only for server connections. Note that GSM modems will also return an IDLE state while waiting for a call in order to show signal strength.
ANSWER	The modem is answering a call and trying to negotiate the Baud rate for the connection. This state appears only for server connections. If the connection is established, the modem will enter the CONNECTED state.
CONNECTED	The modem has established a connection. This state will persist for only a short time, as the LCP negotiation process will begin after a small delay.
NEG LCP	The connection is negotiating LCP options. This process decides on a set of link protocol settings that are acceptable to both the client and the server.
AUTH	The connection is performing the authentication process to ensure that the appropriate user credentials are used.
NEG IPCP	The connection is negotiating IPCP options. This process decides on a set of network protocol settings that are acceptable to both the client and the server.
UP	The connection is active and IP data can be exchanged.
HANGING UP	The modem is disconnecting. This state will exist for only a short time before the modem returns to IDLE.

The signal strength values returned by GSM modems have the following meaning…

VALUE	SIGNAL STRENGTH
0	–113dBm or less.
1	–111dBm.
2-30	–109dBm to –52dBm in 2dBm steps.
31	–51dBm or greater.

VALUE	SIGNAL STRENGTH
99	Signal strength cannot be determined.

Cell phones typically interpret these values as follows when displaying signal strength…

VALUE	STRENGTH	NUMBER OF BARS
5 or less.	–103dBm or less.	One
6 thru 9.	–101dBm thru –95dBm	Two
10 thru 14.	–93dBm thru –85dBm	Three
15 or greater.	–83dBm or greater.	Four

TROUBLESHOOTING MODEM COMMUNICATION

The various modem drivers provide a *Log File* property to log exchange with the modem to a file on the CompactFlash card. This file is used for debugging purpose during initial modem setup or when attempting to find the appropriate configuration options. Be sure to disable this feature once the correct modem configuration sequence has been established.

USING MULTIPLE INTERFACES

Crimson supports up to two modem independent connections. When combined with the one or two Ethernet ports provided by the target device, this gives a total of up to four distinct IP interfaces, all of which will operate according to the configuration parameters defined for each connection. This section describes how these multiple interfaces will interact, and how Crimson will decide where to send each packet of data.

INTERFACE SELECTION

Each interface has an IP address and a network mask, which are used to decide whether to forward packets to that interface. For example, if an Ethernet interface is configured with an IP address of 192.168.1.0 and a network mask of 255.255.255.0, any packets for IP addresses starting with 192.168.1 will be sent to this interface. Likewise, if an on-demand modem connection has a remote IP address of 192.168.2.2 and a network mask of 255.255.255.255, sending a packet to address 192.168.2.2 will result in the connection being established.

Note that this mechanism will only ever send a packet to a single interface. This implies that interfaces should have distinct network addresses, as defined by their IP address ANDed with their network mask. If you breach this requirement, packets will not get routed to the second interface with that network address, and communications on that port will fail. For example, you must not configure one Ethernet port as 192.168.100.1 and the other as 192.168.100.2, as packets for the 192.168.100.0 network will only be sent to the first port.

DEFAULT ROUTE

In addition, one single interface may also define a default route, which will be used to handle packets that do not specifically match any other interface. The method used to configure the route varies according to the interface type, as shown in the table below…

INTERFACE	TO DEFINE DEFAULT ROUTE
Ethernet	Enter a non-zero value for the *Gateway* property.
Dial-In	Enter 0.0.0.0 for the *Remote Mask*.
Dial-Out	Select Default Gateway for the *Route Type* property

Again, only a single interface may define a default route. For example, an operator panel may be connected to a number of Ethernet devices using an IP address of 192.168.1.0 and a network mask of 255.255.255.0, with no gateway defined. An on-demand modem connection may be configured to access an Internet Service Provider so as to send alarm emails. Its *Route Type* is set to Default Gateway, making it the route for any packets for IP addresses that do not match the network defined for the Ethernet port. The SMTP server is configured as 24.104.0.39, resulting in a dial-out connection when an attempt is made to send a message.

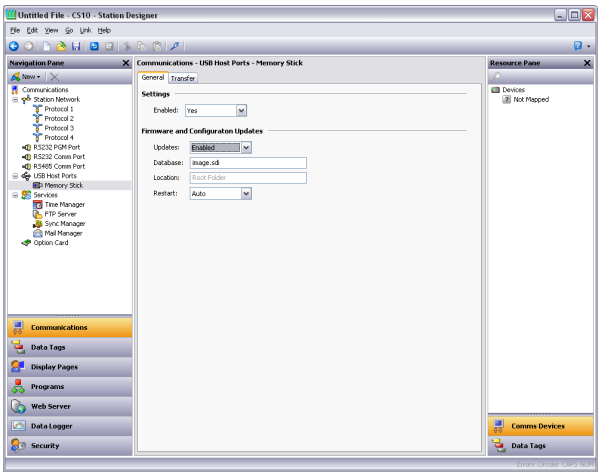
USING THE USB HOST

If your target device has one or more USB host ports, the corresponding icon in the Communications category can be used to configure the devices that it will support. Current builds of Crimson support USB memory devices and keyboards, with the latter category including the many USB bar-code readers that provide keyboard emulation.

MEMORY STICK SUPPORT

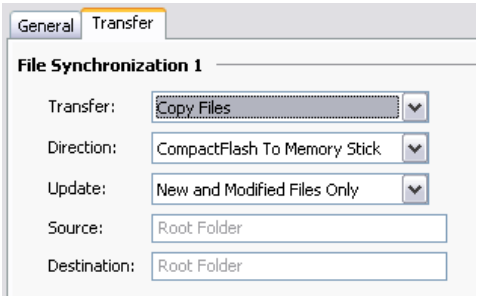
USB memory devices are configured via the Memory Stick icon…

GENERAL PROPERTIES



- The *Enable* property is used to globally disable or enable memory stick support.
- The *Updates* property is used to configure the automatic transfer of new firmware and database to the root directory of the CompactFlash card.
- The *Database* property defines the name of the database image to be copied to the `image.ci3` file on the CompactFlash card. This setting allows several files to be placed on a single stick, with each Crimson device copying the file that is appropriate to its own application.
- The *Location* property specifies the location on the memory stick where the database image file specified above can be located.
- The *Restart* property is used to indicate whether an automatic restart should be performed once the file has been copied. Enabling this property allows the information from the database image to be immediately loaded by Crimson.

TRANSFER PROPERTIES



- The *Transfer* property for each synchronization group defines the function that should be performed. Information may either be copied or moved, and the operation may either be applied simply to the files in the specified folder, or additionally to sub-folders and their contents on a recursive basis.
- The *Direction* property specifies the direction of the transfer.
- The *Update* property is used to indicate whether files that appear to be already present on the target device should be copied in any case, or whether only new and modified files should be transferred. Crimson uses the file's time-stamp and size to decide whether the file should be processed.
- The *Source* and *Destination* properties are used to indicate the folders on the source and target devices where the files should be located.

KEYBOARD SUPPORT

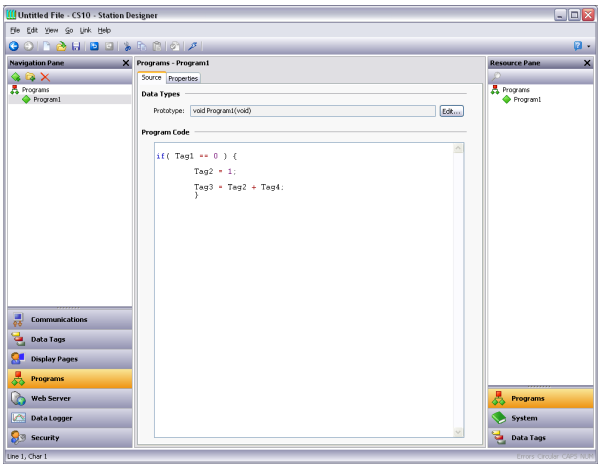
A USB keyboard may be connected to the target device so as to provide a further method of data entry. Crimson primitives will accept the characters from the keyboard as if they had been entered via the popup keypad.

The keyboard is configured via the Keyboard section of the USB Host Ports, with the only option being to choose the keyboard layout so that Crimson can convert the scan-codes into the corresponding characters. Only US and UK layouts are currently supported.

Note that USB barcode readers that implement the keyboard HID class and thereby emulated keyboards are also supported. This allows barcodes to be entered directly into Crimson string tags by means of a simple data entry primitive.

USING PROGRAMS

The previous chapters of this manual refer to using actions to perform operations in response to key or touch-screen presses, or to changes in data tags. If you need to perform an action that is too complex to fit on a single line, or that demands more complex decision-making logic, you can use the Programming category to create and manipulate programs.



THE PROGRAM LIST

The program list in the Navigation Pane is a conventional Navigation List that can be used to create, delete, rename and otherwise organize programs. Note that programs can be grouped into folders, and that each program's icon can display three states: green, indicating a program that has been translated and validated; yellow, indicating a program that has been edited but not yet translated; or red, indicating a program that contains one or more errors.

FINDING PROGRAM USAGE

You can find all the items that refer to a given program by right-clicking that item in the Navigation Pane and selecting the Find Usage command. The resulting items will be placed on the Global Search Results List, and can be accessed by means of the **F4** and **SHIFT+F4** key combinations. The list itself can be shown or hidden by pressing **F8**.

EDITING PROGRAMS

To edit a program, simply edit the program text using the Source tab shown in the Editing Pane. You will notice that the program's icon turns yellow as soon as you start typing, indicating that you have made changes that have yet to be translated. You will also notice that Crimson's program editor performs syntax coloring, auto-indentation and a variety of other features appropriate for a code editor. Editor options can be configured by right-clicking on the Editing Pane and selecting the appropriate command from the resulting menu.

When you have finished writing your program, press the **CTRL+T** key combination or select the Translate button on the toolbar. The program will then be checked for errors. If an error is found, a dialog box will be displayed and the program's icon will turn red. The cursor will

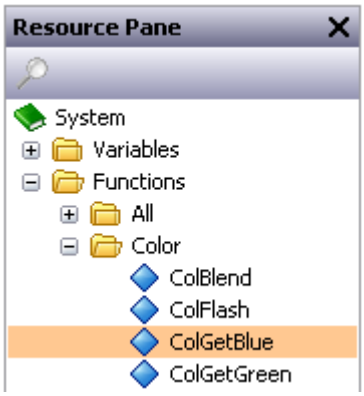
also be moved to the position of the error. If no errors exist, a chime sound will be omitted and the program's icon will turn green, indicating that the program has been translated into a form suitable for execution within the target device.

GETTING HELP

While working within the editing pane, a shortcut is available to provide help on system functions. Place your cursor within or at the end of the function name, and press the **F1** key to display information on the function's operation, arguments and return type. You may also press **F1** after typing a function's name to gain access to the same information.

THE RESOURCE PANE

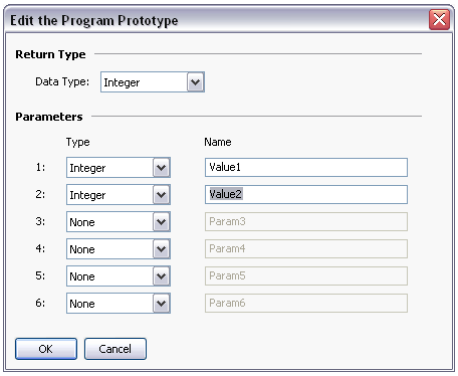
The Resource Pane displayed by the program editor contains a variety of items that can be dragged into your code. The Data Tags and Programs categories are self-explanatory and provide quick access to the respective items in your database by allowing the name of the item to be inserted into the editor. The System category provides access to Crimson's extensive library of system variables and functions…



As you can see, variables and functions are grouped into categories. When a function is selected, its return type and argument types are shown on the status bar. Dropping a function into your code enters the appropriate text, and places the text cursor in the parentheses following the function name, thereby allowing you to enter the required arguments.

PROGRAM DATA TYPES

The field above the program editor can be used to edit the program's data types…

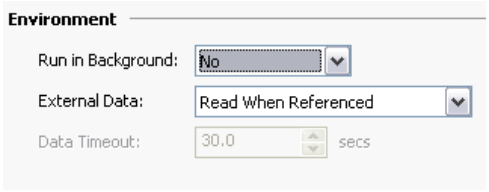


- The *Data Type* property is used to indicate whether this program should simply perform a series of actions, or whether it will perform a calculation and return the value of that calculation to the caller. Programs that return values cannot by definition be run in the background.
- The *Parameters* property section defines up to six parameters that the program will accept. Each parameter has a name and a data type. In this example, the program accepts two parameters, the first named `Value1` and the second named `Value2`, and both being 32-bit signed integers.

Returning values and passing parameters is discussed in more detail below.

PROGRAM PROPERTIES

The second tab of the editor defines the program's execution environment…



- The *Run in Background* property is used to indicate whether Crimson should wait for the program to complete execution before continuing with processing whatever task invoked the program. For example, if this property is set to No, running a program in response to a key being pressed will result in a pause in display updates until the program completes. (Since most programs take very little time to execute, this may not even be noticeable.) If this property is set to Yes, display updates will continue immediately, and the program will execute at a lower priority in the background. Only one background program will run at once, so subsequent requests are queued for later execution. Note also that programs that return values cannot be run in the background, as their return value would then not be available for the caller to use!
- The *External Data* and *Timeout* properties are used to control how the program interacts with Crimson's communication infrastructure with respect to external data items to which the

program refers. You will recall that Crimson only reads data items when they are used. This property is used to control the exact interpretation of this rule with respect to programs…

MODE	BEHAVIOR
Read When Referenced	External data used by the program will be added to the comms scan whenever the program is referenced. If the program is referenced by a display page, the data will be read when that page is displayed; if the program is referenced by a global action or a trigger, the data will be read at all times. This is the default mode, and is acceptable for all programs, except those that use very large amounts of external data.
Read Always	External data used by the program will be read at all times, whether or not the program is referenced. This means that the program will always be ready to run, and that the operator will not see the "NOT READY" message that might otherwise occur when the program is first referenced. The downside of this mode is that comms performance may be reduced if large amounts of data are referenced by the program.
Read When Executed	External data used within the program will be read only when the program is invoked. The program will wait for the period defined in the timeout property for such data to be available. If the data cannot be read—perhaps because a device is offline—the program will not execute. This mode is typically used with globally-referenced programs that consume large amounts of data that would otherwise slow down the communications scan.
Read But Run Anyway	External data will be treated as described for Read Always mode, but the program will execute whether or not the data has been read successfully. The operator will therefore never see the "NOT READY" message, but if a device is offline, there is no guarantee that the program's data items contain valid data.

ADDING COMMENTS

You can add comments to your programs in two ways. First, you can use the `//` sequence to introduce a comment which will continue for the rest of the current line. Secondly, you can use the `/*` sequence to introduce a single- or multi-line comment. This comment will continue until the `*/` sequence appears. The sample below shows both commenting styles…

```
// This is a single-line comment

/* This is line 1 of the comment
   This is line 2 of the comment
   This is line 3 of the comment */
```

A single-line comment may also be placed at the end of a line that contains code.

RETURNING VALUES

As mentioned above, programs can return values. Such programs can be invoked by other programs or by expressions anywhere in the database. For example, if you want to perform a particularly complex decode on a number of conditions relating to a motor and return a value to indicate the current state, you could create a program that returns an integer like this…

```
if( MotorRunning )
       return 1;
else {
       if( MotorTooHot )
              return 2;
       if( MotorTooCold )
              return 3;
       return 0;
       }
```

You could then configure a tag to invoke this program, and use a multi-state format to provide names for the various states. The invocation would be performed by setting the tag's Value property to `Program()`, where `Program` is the name of the program in question. The parentheses are used to indicate a function call and cannot be omitted.

HERE BE DRAGONS!

Note that you have to exercise a degree of caution when using programs to return values. In particular, you should avoid looping for long periods of time, or performing actions that make no sense in the context in which the function will be invoked. For example, if the code fragment above called the `GotoPage` function to change the page, the display would change every time the program was invoked. Imagine what would happen if you, say, tried to log data from the associated tag, and you'll realize that this would not be a good thing! Therefore, keep programs that return values simple, and always consider the context in which they will be run. If in doubt, avoid doing anything other than simple math and `if` statements.

PASSING ARGUMENTS

As mentioned above, programs can accept arguments. Suppose you want to write a program called `FindMean` to take the average of two integer values. The program would be configured to accept two integer arguments, `a` and `b`. The program would also be configured so as to return an integer. The code within the program would then be defined as…

```
return (a+b)/2;
```

Once this program has been created and translated, you will be able to enter an expression such as `FindMean(Tag1, Tag2)` to invoke it with the appropriate arguments. In this case, the expression would be equal to the average of `Tag1` and `Tag2`.

PROGRAMMING TIPS

The sections below provide an overview of the programming constructions supported by Crimson. The basic syntax used is that of the C programming language. Note that the aim is not to try to teach you to become a programmer, or to master the subtleties of the C language. Such topics are beyond the scope of this manual. Rather, the aim is to provide a quick overview of the facilities available, so that the interested user might experiment further.

MULTIPLE ACTIONS

The simplest type of program comprises a list of actions, with each action taking up a single line, and being followed by a semicolon. All of the various actions defined in the Writing Actions section are available for use. Simple programs like this are typically used where combining the actions in a single action definition would otherwise prove unreadable.

The sample shown below sets several variables, and then changes the display page…

```
Motor1 = 0;
Motor2 = 1;
Motor3 = 0;

GotoPage(Page1);
```

The actions will be executed in order, and the program will then return to the caller.

IF STATEMENTS

This type of statement is used within a program to make a decision. The construct consists of an `if` statement with a condition in parentheses, followed by an action (or actions) to be executed if the condition is true. If more than one action is specified, each should be placed on a separate line, and curly-brackets should be used to group the statements together. An optional `else` clause can be used to provide for code to be run if the condition is false.

The example below shows an `if` statement with a single action…

```
if( TankFull )
       StartPump = 1;
```

The example below shows an `if` statement with two actions…

```
if( TankEmpty ) {
       StartPump = 0;
       OpenValue = 1;
       }
```

The example below shows an `if` statement with an `else` clause…

```
if( MotorHot )
       StartFan = 1;
else
       StartFan = 0;
```

Note that it is very important to remember to place the curly-brackets around groups of actions to be executed in the `if` or `else` portion of the statement. If you omit the brackets, Crimson will most likely misunderstand exactly which actions you want to be dependent upon the `if` condition. Although line breaks are recommend between actions, they are not used to figure out what is and is not included within the conditional statement.

SWITCH STATEMENTS

A `switch` statement is used to compare an integer value against a number of possible constants, and to perform an action based upon which value is matched. The exact syntax supports a number of options beyond those shown in the example below, but for the vast majority of applications, this simple form will be acceptable.

This example below will start a motor selected by the value in the `MotorIndex` tag...

```
switch( MotorIndex ) {

       case 1:
              MotorA = 1;
              break;
       case 2:
       case 3:
              MotorB = 1;
              break;
       case 4:
              MotorC = 1;
              break;
       default:
              MotorD = 1;
              break;
       }
```

A value of 1 will start motor A, a value of 2 or 3 will start motor B, and a value of 4 will start motor C. Any value which is not explicitly listed will start motor D. Things to note about the syntax are the use of curly-brackets around the `case` statements, the use of `break` to end each conditional block, the use of two sequential `case` statements to match more than one value, and the use of the optional `default` statement to indicate an action to perform if none of the specified values is matched by the value in the controlling expression. (If this syntax looks too intimidating, a series of `if` statements can be used instead to produce the same results, but with marginally lower performance, and somewhat less readability.)

LOCAL VARIABLES

Some programs use variables to store intermediate results, or to control one of the various loop constructs described below. Rather than defining a tag to hold these values, you can declare what are known as local variables using the syntax shown below…

```
int     a;          // Declare local integer 'a'
float   b;          // Declare local real    'b'
cstring c;          // Declare local string  'c'
```

Local variables may optionally be initialized when they are declared by following the variable name with = and the value to be assigned. Variables that are not initialized in this manner are set to zero, or an empty string, as appropriate.

Note that local variables are truly local in both scope and lifetime. This means that they cannot be referenced outside the program, and they do not retain their values between function invocations. If a function is called recursively, each invocation has its own variables.

LOOP CONSTRUCTS

The three different loop constructs can be used to perform a given section of code while a certain condition is true. The `while` loop tests its condition before the code is executed, while the `do` loop tests the condition afterwards. The `for` loop is a quicker way of defining a `while` loop, allowing you to combine three common elements into one statement.

You should note that some care is required when using loops within your programs, as you may make a programming error which results in a loop that never terminates. Depending on the situation in which the program is invoked, this may seriously disrupt the terminal's user interface activity, or its communications. Loops which iterate too many times may also cause performance issues for the subsystem that invokes them.

THE WHILE LOOP

This type of loop repeats the action that follows it while the condition in the `while` statement remains true. If the condition is never true, the action will never be executed, and the loop will perform no operation beyond evaluating the controlling condition. If you want more than one action to be included in the loop, be sure to surround the multiple statements in curly-brackets, as with the `if` statement. The example below initializes a pair of local variables, and then uses the first to loop through the contents of an array, totaling the first ten elements, and returning the total value to the caller…

```
int i=0, t=0;

while( i < 10 ) {
       t = t + Data[i];
       i = i + 1;
       }

return t;
```

The example below shows the same program, but rewritten in a compressed form. Since the loop statement now controls only a single action, the curly-brackets have been omitted…

```
int i=0, t=0;

while( i < 10 )
       t += Data[i++];

return t;
```

THE FOR LOOP

You will notice that the `while` loop shown above has four elements…

1. The initialization of the loop control variable.
2. The evaluation of a test to see if the loop should continue.
3. The execution of the action to be performed by the loop.
4. The making of a change to the control variable.

The `for` loop allows elements 1, 2 and 4 to be combined within a single statement, such that the action following the statement need only implement element 3. This syntax results in something similar to the FOR-NEXT loop found in BASIC and other such languages.

Using this statement, the example given above can be rewritten as…

```
int i, t;

for( i=t=0; i<10; i++ )
       t += Data[i];

return t;
```

You will notice that the `for` statement contains three distinct elements, each separated by semicolons. The first element is the initialization step, which is performed once when the loop first begins; the next is the condition, which is tested at the start of each loop iteration to see if the loop should continue; the final element is the induction step, which is used to make a change to the control variable to move the loop on to its next iteration. Again, if you want more than one action to be included in the loop, include them in curly-brackets!

THE DO LOOP

This type of loop is similar to the `while` loop, except that the condition is tested at the end of the loop. This means that the loop will always execute at least once.

The example below shows the example from above, rewritten to use a `do` loop…

```
int i=0, t=0;

do {
       t += Data[i];
       } while( ++i < 10 );

return t;
```

LOOP CONTROL

Two additional statements can be used within loops. The `break` statement can be used to terminate the loop early, while the `continue` statement can be used to skip the balance of the loop body and begin another iteration without executing any further code. To make any sense, these statements must be used with `if` statements to make their execution conditional.

The example below shows a loop that terminates early if another program returns true…

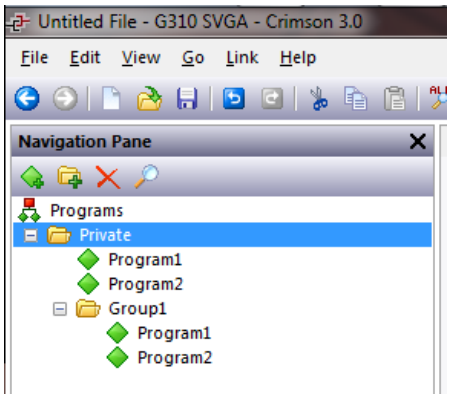
```
for( i=0; i<10; i++ ) {
       if( LoopAbort() )
              break;
       LoopBody();
       }
```

PASSWORD PROTECTING PROGRAMS

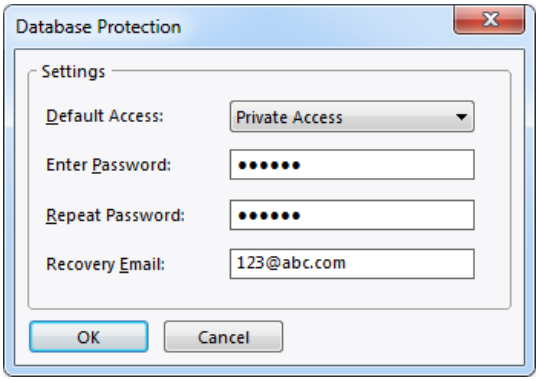
Programs can be password protected and hidden from view by placing them in a folder named "Private".

PROTECTING PROGRAMS

Create a single folder named "Private" and place all programs to be protected into this folder. Subfolders under the "Private" folder can be created and named as desired and they will be protected as well.

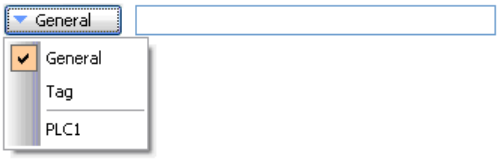


On the menu bar, select "File" then "Protection." In the dialog box, select "Private Access" and then enter the required information. After protection is complete, the "Private" folder can be seen, but programs will be hidden. To unprotect programs, select "File" then "Protection." Enter the password and then select "Full Access" in the dropdown menu.



WRITING EXPRESSIONS

You will recall from the earlier sections of this manual that many fields within Crimson are configured as what are called expression properties. You will further recall that these fields are edited by means of a user interface element similar to that shown below…



In many situations, you will be configuring these properties to be equal to the value of a tag, or to the contents of a register in a remote communications device. In these cases, you will either be dragging items from the Resource Pane, or you will be clicking the appropriate option on the drop-down menu, and then selecting the item from the resulting dialog box.

There will be situations, though, when you want to make a property dependent on a more complex combination of data items, perhaps using some math to combine or compare their values. Such eventualities are handled via what are known as expressions, which can be entered in the property's edit box whenever General mode is selected via the drop-down.

DATA VALUES

All expressions contain at least one data value. The simplest expressions are thus references to single constants, single tags or single PLC registers. If you enter either of the last two options, Crimson will simplify the editing process by automatically changing the property mode as appropriate. For example, if you enter a tag name in General mode, Crimson will switch to Tag mode, and show the tag name in the selection field.

CONSTANTS

Constants represent—not surprisingly—constant numbers or strings.

INTEGER CONSTANTS

Integer constants represent a single 32-bit signed number. They may be entered in decimal, binary, octal or hexadecimal as required. The examples below show the same number entered in the four different number bases…

BASE	EXAMPLE
Decimal	`123`
Binary	`0b1111011`
Octal	`0173`
Hexadecimal	`0x7B`

The 'U' and 'L' suffixes supported by earlier versions of software are not used.

CHARACTER CONSTANTS

Character constants represent a single Unicode character, encoded in the lower 16 bits of a 32-bit signed number. A character constant comprises a single character enclosed in single quotation marks, such that `'A'` can be used to represent a value of 65. Certain otherwise unprintable or unrepresentable characters can be encoded using what are called escape sequences, each of which is introduced with a single backslash…

SEQUENCE	VALUE	ASCII
\a	Hex 0x07, Decimal 7	BEL
\t	Hex 0x09, Decimal 9	TAB
\n	Hex 0x0A, Decimal 10	LF
\f	Hex 0x0C, Decimal 12	FF
\r	Hex 0x0D, Decimal 13	CR
\e	Hex 0x1B, Decimal 27	ESC
\x*nn*	The hex value represented by *nn*.	-
\u*nnnn*	The hex value represented by *nnnn*.	-
*****nnn*	The octal value represented by *nnn*.	-
****	A single backslash character.	-
\'	A single quotation mark character.	-
\"	A double quotation mark character.	-

LOGICAL CONSTANTS

Logical constants represent a 1 or 0 value that is used to indicate the truth or otherwise of a yes-or-no expression. An example of something that can be assigned to be equal to a logical constant is a tag that represents a digital output in a PLC. Logical constants can either be entered simply as `1` or `0`, or by use of the keywords `true` or `false`.

FLOATING-POINT CONSTANTS

Floating-point constants represent a 32-bit single-precision floating point value. They are represented as you might expect—by the integer portion, followed by a single decimal point, followed by the fractional portion. Scientific notation is also supported by specifying a value for the mantissa and following this with an ‘E’ and an exponent.

STRING CONSTANTS

String constants represent sequences of characters. They comprise the characters to be represented, enclosed in double quotation marks. For example, the string `"ABCD"` represents a four-character string, comprising the values 65, 66, 67 and 68. (Actually, five 16-bit words are used to store the string, with a null value being appended as a terminator.) The various escape sequences discussed above may also be used within strings.

TAG VALUES

The value of a tag is represented in an expression by the tag name. Tags that are organized into folders are represented by the pathname of the tag with each pair of elements being separated by a period. A tag named PV in a folder named Loop would thus be referenced as `Loop.PV`. Note that upper-case and lower-case characters are considered equivalent when finding the required tag. Once an expression has been entered, any changes to the name of the tag will modify all of the expressions that make reference to it.

TAG PROPERTIES

Data tags have certain properties than can be accessed by following a tag name with a period and then with the name of the required property. The following properties are defined…

PROPERTY	DESCRIPTION	DATA TYPE
`Name`	The tag's name.	String.
`AsText`	The tag's value formatted as text.	String.
`Label`	The tag's Label property.	String.
`Desc`	The tag's Description property.	String.
`Prefix`	The prefix defined by the tag's format.	String.
`Units`	The units defined by the tag's format.	String.
`SP`	The tag's setpoint property.	Same As Tag.
`Min`	The tag's lower data entry limit.	Same As Tag.
`Max`	The tag's upper data entry limit.	Same As Tag.
`Fore`	The tag's current foreground color.	Integer.
`Back`	The tag's current background color.	Integer.

PAGE PROPERTIES

Display pages also have certain properties that can be accessed in the same way…

PROPERTY	DESCRIPTION	DATA TYPE
`Name`	The page's name.	String.
`Label`	The page's Label property.	String.
`Desc`	The page's Description property.	String.

COMMS REFERENCES

References to registers in master communications devices can be entered into an expression by means of a syntax comprising an opening square bracket, the register name, and a closing square bracket. An optional device name may be prefixed to the register name and separated by a period. The device name is not needed when referring to the only device in a database.

Examples of this syntax are shown below…

EXAMPLE	MEANING
`[D100]`	Register **D100** in first device.
`[AB.N7:0]`	Register **N7:0** in device **AB**.
`[FX.D100]`	Register **D100** in device **FX**.

SIMPLE MATH

As mentioned above, expressions often contain more than one data value, with their values being combined mathematically. The simplest of these expressions may add a pair of values, while a more complex expression might obtain the average of three values. These operations are performed using the familiar syntax you will have seen in applications such as Excel. The examples below show the basic operations that can be performed…

OPERATOR	PRIORITY	EXAMPLE
Addition	Group 4	**Tag1 + Tag2**
Subtraction	Group 4	**Tag1 - Tag2**
Multiplication	Group 3	**Tag1 * Tag2**
Division	Group 3	**Tag1 / Tag2**
Remainder	Group 3	**Tag1 % Tag2**

Although the examples show spaces surrounding the operators, these are not required.

OPERATOR PRIORITY

You will have noticed the Priority column in the above table. As you no doubt recall from your algebra classes, when several operators are used together, they are evaluated in a defined order. For example, multiplication is always evaluated before addition. Crimson implements this ordering by means of what are known as operator priorities, with each operator being placed in a group, and with operators being applied from the lowest numbered group to the highest. Except where noted otherwise in the text, operators within a group are evaluated left-to-right. The default order of evaluation can be overridden by using parentheses.

TYPE CONVERSION

Normally, Crimson will automatically decide when to switch from evaluating an expression in integer math to evaluating it using floating point. For example, if you divide an integer value by a floating point value, the integer will be converted to floating point before the division is carried out. However, there will be some situations where you want to force a conversion to take place.

For example, suppose you are adding together three integers that represent the levels in three tanks, and then dividing the total by the tank count to obtain the average level. If you use an expression such as `(Tank1+Tank2+Tank3)/3` then your result may not be as accurate as you demand, as the division will take place using integer math, and the average will not contain any decimal places. To force Crimson to evaluate the result using floating-point math, the simplest technique is to change the `3` to `3.0`, thereby forcing Crimson to convert the sum to

floating point before the division is performed. A slightly more complex technique is to use syntax such as `float(Tank1+Tank2+Tank3)/3`. This invokes what is known as a type cast on the term in parentheses, manually converting it to floating point.

Type casts may also be used to convert a floating-point value to an integer value, perhaps deliberately giving-up some precision from an intermediate value before storing it in a PLC register. For example, the expression `int(cos(Theta)*100)` will calculate the cosine of an angle, multiply this value by 100 using floating-point math, before converting it to an integer, dropping any digits after the decimal place.

COMPARING VALUES

You will quite often find that you wish to compare the value of one data with another, and make a decision based on the result. For example, you may wish to define a flag formula to show when a tank exceeds a particular value, or you may wish to use an `if` statement in a program to execute some code when a motor reaches its desired speed. The following comparison operators are provided…

OPERATOR	PRIORITY	EXAMPLE
Equal To	Group 7	`Data == 100`
Not Equal To	Group 7	`Data != 100`
Greater Than	Group 6	`Data > 100`
Greater Than or Equal To	Group 6	`Data >= 100`
Less Than	Group 6	`Data < 100`
Less Than or Equal To	Group 6	`Data <= 100`

Each operator produces a value of 0 or 1, depending on the condition it tests. The operators can be used on integers, floating point values or strings. If strings are being compared, the comparison is case-insensitive such that “abc” is considered equal to “ABC”.

TESTING BITS

Crimson allows you to test the value of a bit within a data value by using the bit selection operator, which is represented by a single period. The left-hand side of the operator should be the value in which the bit is to be tested, and the right-hand side should be an expression indicating the bit number to test. This right-hand value should be between 0 and 31. The result of the operator is equal to 0 or 1 depending on the value of the bit in question.

OPERATOR	PRIORITY	EXAMPLE
Bit Selection	Group 1	`Input.2`

The example shown tests bit 2 (i.e. the bit with a value of 4) within the indicated tag.

If you want to test for a bit being equal to zero, you can use the logical NOT operator…

OPERATOR	PRIORITY	EXAMPLE
Logical NOT	Group 2	`!Input.2`

This example is equal to 1 if bit 2 of the indicated tag is equal to 0, and vice versa.

MULTIPLE CONDITIONS

If you want to define an expression that is true if a number of conditions are *all* true, you can use the logical AND operator. Similarly, if you want to define an expression that is true if *any* of a number of conditions are true, you can use the logical OR operator. The examples below show each operator in use…

OPERATOR	PRIORITY	EXAMPLE
Logical AND	Group 11	`A>10 && B>10`
Logical OR	Group 12	`A>10 || B>10`

The logical AND operator produces a value of 1 if and only if the expressions on the left-hand and right-hand sides are true, while the logical OR operator produces a value of 1 if either expression is true. Note that—unlike the bitwise operators referred to elsewhere in this section—the logical operators stop evaluating once they know what the answer will be. This means that in the above example for logical AND, the right-hand side of the operator will only be evaluated if A is greater than 10, as, if this were not true, the result of the AND operator must already be zero. While this property makes little difference in the examples given above, if the left-hand or right-hand expressions call a program or make a change to a data value, this behavior must be taken into account.

CHOOSING VALUES

You may find situations where you want to select between two values—be they integers, floating point values or strings—depending on the value of some condition. For example, you may wish to set a motor's speed equal to 500 rpm or 2000 rpm based on a flag tag. This operation can be performed using the `?:` operator, which is unique in that it takes three arguments, as shown in the example below…

OPERATOR	PRIORITY	EXAMPLE
Selection	Group 13	`Fast ? 2000 : 500`

This example will evaluate to 2000 if `Fast` is true, and 500 otherwise. The operator can be thought to be equivalent to the `IF` function found in applications such as Microsoft Excel.

MANIPULATING BITS

Crimson also provides operators to perform operations that do not treat integers as numeric values, but instead as sequences of bits. These operators are known as bitwise operators.

AND, OR AND XOR

These three bitwise operators each produce a result in which each bit is defined to be equal to the corresponding bits in the values on the operator's left-hand and right-hand sides, combined using a specific truth-table…

OPERATOR	PRIORITY	EXAMPLE
Bitwise AND	Group 8	`Data & Mask`
Bitwise OR	Group 9	`Data | Mask`
Bitwise XOR	Group 10	`Data ^ Mask`

The table below shows the associated truth tables…

A	B	A & B	A | B	A ^ B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

SHIFT OPERATORS

Crimson also provides operators to shift an integer *n* bits to the left or right…

OPERATOR	PRIORITY	EXAMPLE
Shift Left	Group 5	`Data << 2`
Shift Right	Group 5	`Data >> 2`

Each example shifts `Data` two bits in the specified direction.

BITWISE NOT

Finally, Crimson provides a bitwise NOT operator to invert the sense of the bits in a value…

OPERATOR	PRIORITY	EXAMPLE
Bitwise NOT	Group 2	`~Mask`

This example produces a value where every bit is equal to the opposite of its value in `Mask`.

INDEXING ARRAYS

Elements within an array tag can be selected by following the array name with square brackets that contain an indexing expression. This expression must range from 0 to one less than the number of elements in the array. If you create a 10-element array, for example, the first element will be `Name[0]` and the last will be `Name[9]`.

INDEXING STRINGS

Square brackets can also be used to select characters within a string. For example, if you have a tag called Text that contains the string "ABCD", then the expression `Text[0]` will return a value of 65, this being equal to the Unicode value of the first character. Index values beyond the end of the string will always return zero.

ADDING STRINGS

As well as adding numbers, the addition operator can be used to concatenate strings. Thus, the expression **`"AB"+"CD"`** evaluates to "ABCD". You may also use the addition operator to add an integer to a string, in which case a single character equal to the Unicode representation of the integer is appended to the data in the string.

CALLING PROGRAMS

Programs that return values may be invoked within expressions by following the program name with a pair of parentheses. For example, `Program1()*10` will invoke the associated program, and multiply the return value by 10. Obviously, the return type for `Program1` must be set to integer or floating point for this to make sense.

USING FUNCTIONS

Crimson provides a number of predefined functions that can be used to access system information, or to perform common math operations. These functions are defined in detail in the Function Reference. They are invoked using a syntax similar to that for programs, with any arguments to the function being enclosed within the parentheses. For example, `cos(0)` will invoke the cosine function with an argument of 0, returning a value of +1.0.

PRIORITY SUMMARY

The table below shows the priority of all the operators defined in this section…

GROUP	OPERATORS
Group 1	**.**
Group 2	**! ~**
Group 3	*** / %**
Group 4	**+ -**
Group 5	**<< >>**
Group 6	**< > <= >=**
Group 7	**== !=**
Group 8	**&**
Group 9	**|**
Group 10	**^**
Group 11	**&&**
Group 12	**||**

GROUP	OPERATORS
Group 13	? :

Operators in the lower-numbered groups are applied first.

WRITING ACTIONS

While expressions define values, actions define what you want to happen when an event occurs. The vast majority of the actions in a database will relate to interactions with primitives or with the keyboard. Since Crimson provides a simple method of defining commonly-used actions for these items, you will often be able to avoid writing actions by hand. Actions are needed, though, if you want to use triggers, write programs, or use a key or primitive in User Defined mode.

CHANGING PAGE

To create an action that changes the page shown on the panel's display, use the syntax `GotoPage(Name)`, where `Name` is the name of the display page in question. The current page will be removed, and the new page will be displayed in its place.

CHANGING NUMERIC VALUES

Crimson provides several ways of changing data values.

SIMPLE ASSIGNMENT

To create an action that assigns a new value to a tag or to a register in a communications device, use the syntax `Data=Value`, where `Data` is the data item to be changed, and `Value` is the value to be assigned. Note that `Value` need not just be a constant value, but can be any valid expression of the correct type. Refer to the previous section for details of how to write expressions. For example, code such as `[N7:0]=Tank1+Tank2` can be used to add two tank levels and store the total quantity directly in a PLC register.

COMPOUND ASSIGNMENT

To create an action that sets a data value equal to its current value combined with another value by means of any of the operators defined in the previous section, use the syntax `Data`***op***`=Value`, where `Data` is the tag to be changed, `Value` is the value to be used by the operator, and ***op*** is any of the available operators. For example, the code `Tag+=10` will increase `Tag` by a value of 10, while `Tag*=10` will multiply the current value by 10.

INCREMENT AND DECREMENT

To create an action that increases a data value by one, use the syntax `Data++`. To create an action that decreases a tag by one, use the syntax `Data--`. Note that the `++` or `--` operators may be placed before or after the data value in question. In the former case, the value of the expression represented by `++Data` is equal to the value of `Data` *after* it has been incremented. In the latter case, the expression is equal to the value *before* it has changed.

CHANGING BIT VALUES

To change a bit within a tag, use the syntax `Data.Bit=1` or `Data.Bit=0` to set or clear the bit as required, where `Data` is the tag in question and `Bit` is the zero-based bit number. Note

again that the value on the right-hand side of the = operator can be an expression if desired, such that an example such as `Data.1=(Level>10)` can be used to set or clear a bit depending on whether or not a tank level exceeds a preset value.

RUNNING PROGRAMS

Programs may be invoked within actions by following the program name with a pair of parentheses. For example, `Program1()` will invoke the associated program. The program will execute in the foreground or background as defined by the program's properties.

USING FUNCTIONS

Crimson provides a number of predefined functions that can be used to perform various operations. These functions are defined in detail in the Function Reference. They are invoked using a syntax similar to that for programs, with any arguments to the function being enclosed within the parentheses. For example, `SetLanguage(1)` will set the terminal language to 1.

OPERATOR PRIORITY

All assignment operators fall into Group 14. In other words, they will be evaluated after all other operators in an action. They are also unique in that they group right-to-left. This means that code such as `Tag1=Tag2=Tag3=0` can be used to clear all three tags at once.