

Thank you for choosing Red Lion Controls' Paradigm family of operator interfaces. Included with this manual is a copy of EDICT-97. This intuitive based programming software can be used to program the entire line of Paradigm products available from Red Lion Controls. EDICT-97 has many features to ensure that you have all the power you need for your application. These powerful facilities are available to you with unrivaled ease-of-use.



**Red Lion Controls Inc**

20 Willow Springs Circle, York, PA 17402

Tel +1 (717) 767-6511 Fax: +1 (717) 764-6587

**Red Lion UK Ltd**

Tapton Park, Chesterfield, Derbyshire S41 0TZ

Tel +44 (1246) 22 21 22 Fax: +44 (1246) 22 12 22

**Red Lion Controls France**

56 Boulevard du Courcerin Batiment 21, ZI Pariest

F-77183 Croissy Beaubourg, France

Tel +33 (64) 80 12 12 Fax: +33 (64) 80 12 13

[www.redlion-controls.com](http://www.redlion-controls.com)

[sales@redlion-controls.com](mailto:sales@redlion-controls.com)

**© 2000, RED LION CONTROLS, INC. ALL RIGHTS RESERVED.**

Information in this document is subject to change without notice and does not represent a commitment by Red Lion Controls. Software, which includes any database supplied therewith, described in this document may be furnished subject to a license agreement or a nondisclosure agreement. It is against the Law to copy the software except as specifically allowed in the license or nondisclosure agreement. No part of this document may be reproduced in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose, without the express written permission of Red Lion Controls.

PowerPoint and Windows are registered trademarks of Microsoft Corporation. Other product and company names mentioned herein may be the trademarks of their respective owners.

***Disclaimer***

Red Lion Controls, hereinafter referred to as RLC, will under no circumstances be responsible for direct, indirect, special, incidental or consequential damages, death or personal injury arising from the use or misuse of all or part of this documentation or the products and software described herein. Notwithstanding the above, RLC does not exclude any liability for death or personal injury caused by its negligence.

RLC does not warrant any of its software products to be free from error or to be fit for any particular purpose. Neither is the software guaranteed to provide operation without interruption. The customer's sole remedy in case of failure is the refund of the purchase price of the software. The customer, in applying the products and software described herein, accepts that the products are wholly or partly programmable electronic systems that are inherently complex and which cannot thus be guaranteed to be free of errors. In doing so, the customer accepts the responsibility to ensure that the products are correctly programmed, configured, installed, commissioned, operated and maintained by competent and suitably trained staff and according to any instructions or safety instructions provided and as dictated by good engineering practices. This documentation, and the software and products described herein, is subject to continuous development and improvement. All information is given in good faith, but RLC shall not be liable for any omissions or errors herein or within the software herein described.

***Contact Details***

Red Lion Controls Inc  
20 Willow Springs Circle  
York  
PA 17402  
U.S.A.  
Tel +1 (717) 767-6511  
Fax +1 (717) 764-6587

# TABLE OF CONTENTS

<b>HARDWARE SPECIFICATIONS .....</b>	<b>5</b>
Power Supply Requirements .....	21
Changing the Battery.....	21
Function Key Strips .....	21
<b>GETTING STARTED .....</b>	<b>23</b>
Installing EDICT-97 .....	23
A Quick Start of EDICT-97 .....	37
<b>THE EDICT-97 DATABASE CONTENTS CATEGORIES .....</b>	<b>38</b>
Display Pages .....	41
Insert Integer Value.....	42
Direct PLC References .....	45
Display Page Properties.....	50
Data Entry Properties.....	52
Using Events and Actions .....	52
Inserting Animation Items .....	56
Bitmaps (Graphic Units Only) .....	64
Character Fonts (Graphic Units Only).....	67
Global Events.....	70
SoftKey Menus.....	71
Security System .....	73
Comms Ports .....	77
Comms Devices .....	78
Comms Blocks .....	79
Named Data .....	80
Using Named Data For Recipes .....	81
Importing CSV Files .....	84
Alarm Scanner .....	86
Trigger Table .....	89
Schedule Table .....	90
Selection Table .....	91
Message Table .....	93
Data Logger (Graphic Units Only).....	93
Event Logs .....	96
Printed Reports .....	99
User Programs .....	99
Multiple Language Feature .....	100

## SECTION A - FUNCTIONS

<b>USING FUNCTIONS.....</b>	<b>A1</b>
Index of Functions .....	A1
The Compound Statement .....	A43
If-Else Statement .....	A43
Loop Statements .....	A44
The While Loop .....	A45

The Do-While Loop .....	A45
The For Loop .....	A45
The Switch Statement .....	A46
The Return Statement .....	A46

## **SECTION B - USER PROGRAMS/OPERATORS/SYSTEM VARIABLES**

<b>PART 1 - USING PROGRAMS</b> .....	B1
Simple Programs .....	B1
Complex Programs .....	B1
Writing Programs .....	B2
Functions .....	B2
Statement Types .....	B2
The Action Statement .....	B2
The Compound Statement .....	B2
Using Actions .....	B6
Modifying Data .....	B6
Modifying Bits .....	B6
Using Functions .....	B7
Using Expressions .....	B7
Comments .....	B8
Type Names .....	B8
Data Types .....	B8
Type Casting .....	B9
Key Words .....	B9
<b>PART 2 - OPERATORS PAGES</b> .....	B11
<b>PART 3 - SYSTEM VARIABLES</b> .....	B20
System Variable Index .....	B20
System Variable Descriptions .....	B21

## **SECTION C - ERROR CODES**

<b>LEXICAL ERROR INDEX</b> .....	C1
<b>COMPILER ERROR INDEX</b> .....	C5

## **SECTION D - DRIVER SELECTIONS**

<b>COMMUNICATIONS</b> .....	D1
Automatic Configuration .....	D1
Direct PLC References .....	D1
Connecting Your HMI to a PLC .....	D1
Accessing a Register in your PLC .....	D1
<b>CONNECTING YOUR HMI TO RED LION CONTROLS PRODUCTS</b> .....	D4
<b>TROUBLESHOOTING COMMUNICATIONS</b> .....	D7
LED's .....	D7
Comms Errors .....	D8
Comms Update .....	D8
<b>THE PARADIGM RS485/422 PORT</b> .....	D10

<b>PARADIGM TO PARADIGM COMMUNICATIONS - PC LINK .....</b>	<b>D11</b>
Configuring a Master .....	D11
Using the Wizard .....	D11
Manual Configuration .....	D11
Configuring the Slaves .....	D13
Using the Wizard .....	D13
Manual Configuration .....	D13
Device Connections for PC Link .....	D15
Using RS232 .....	D15
Using RS422 .....	D15
<b>GENERAL ASCII FRAME.....</b>	<b>D17</b>
<b>EDICT-97 COMMUNICATION DRIVERS .....</b>	<b>D19</b>
<b>PARADIGM CABLE GUIDE .....</b>	<b>D23</b>

## **SECTION E – GRAPHIC UNITS**

<b>THE GRAPHICS ANIMATION TOOLBOX.....</b>	<b>E3</b>
<b>INSERTING ANIMATION ITEMS .....</b>	<b>E6</b>
<b>PowerPoint-STYLE PAGE TRANSITIONS .....</b>	<b>E8</b>
<b>CHOOSING A DISPLAY PAGE COLOR.....</b>	<b>E8</b>
<b>ALIGNING ANIMATION ITEMS.....</b>	<b>E9</b>
<b>GROUPING ANIMATION ITEMS .....</b>	<b>E11</b>
<b>USING GROUP PROPERTIES IN MACROS.....</b>	<b>E14</b>
<b>INSERTING ANIMATION ITEMS FROM EDICT-97's LIBRARY .....</b>	<b>E19</b>
Inserting a Horizontal or Vertical Fill .....	E20
Inserting a Rotary Pointer .....	E24
<b>THE VERTICAL BAR GRAPH ANIMATION ITEM.....</b>	<b>E30</b>
<b>INSERTING LINE GRAPH ANIMATION ITEMS ON A DISPLAY PAGE.....</b>	<b>E34</b>
<b>USING THE DATA LOGGER.....</b>	<b>E38</b>
<b>DISPLAYING A TREND ON THE GRAPHICS LAYER OF A DISPLAY PAGE.....</b>	<b>E39</b>
<b>THE COLOR SELECTION TABLE.....</b>	<b>E41</b>
<b>TO LOAD CUSTOM BITMAPS.....</b>	<b>E46</b>
<b>ANIMATION ITEMS</b>	
Integer Animation Item .....	E49
Real Number Animation Item .....	E50
Fixed Text Animation Item .....	E52
Status Text Animation Item .....	E53
Quad Text Animation Item .....	E54
Message Text Animation Item .....	E56
Decode Text Animation Item .....	E58
General Text Animation Item .....	E59
Time & Date Animation Item .....	E60
The Line Animation Item .....	E61
The Frame Animation Item .....	E62
The Rectangle Animation Item .....	E62
The Shadow Animation Item .....	E63
The Wedge Animation Item .....	E63
The Circle Animation Item .....	E63
The Disk Animation Item .....	E64



## Hardware

### **Pictured Below: CX200** (front panel view)

#### **Display Options**

Vacuum  
Fluorescent or  
Backlit LCD  
  
2 X 20, 4 X 20, or  
2 X 40  
(lines X characters)

User  
legendable  
Function  
Keys

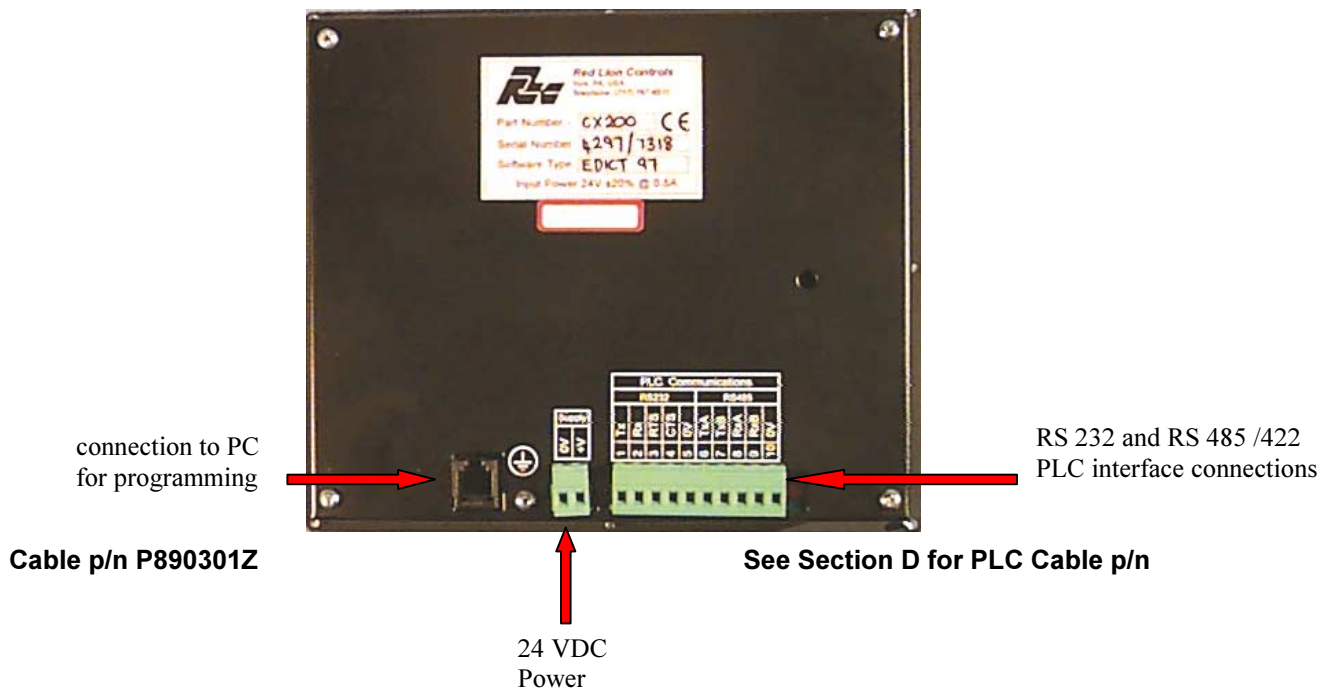


Screen  
legendable  
Soft Keys

Numeric Keypads  
With raise, lower,  
Next, etc.

**All keys feature Tactile Feedback**

### **CX -200** (rear panel view)



connection to PC  
for programming

RS 232 and RS 485 /422  
PLC interface connections

**Cable p/n P890301Z**

**See Section D for PLC Cable p/n**

24 VDC  
Power

## MODEL CL05 - PARADIGM 2 X 20 LCD OPERATOR INTERFACE TERMINAL



- 2 LINE X 20 CHARACTER LIQUID CRYSTAL DISPLAY WITH LED BACKLIGHT
- 100 ALARM POINT LOGGER
- RECIPE HANDLING
- UNLIMITED PASSWORD PROTECTION
- REAL TIME CLOCK, BATTERY BACKED
- EXPRESSION EVALUATION
- 32-BIT MATH
- DIRECT PLC COMMUNICATION
- NEMA 4/IP65 STEEL ENCLOSURE



### DESCRIPTION

The Paradigm operator interface Model CL-05 was designed to meet the industrial demands of application power, versatility, reliability, and ease of use. The CL-05 has provision, common to all Paradigm Family products, allowing for future product upgrades as new options and capabilities are developed.

### SAFETY SUMMARY

All safety related regulations, local codes and instructions that appear in the manual or on equipment must be observed to ensure personal safety and to prevent damage to either the instrument or equipment connected to it. If equipment is used in a manner not specified by the manufacturer, the protection provided by the equipment may be impaired.

Do not use this unit to directly command motors, valves, or other actuators not equipped with safeguards. To do so, can be potentially harmful to persons or equipment in the event of a fault to the unit.



**CAUTION:** Read complete instructions prior to installation and operation of the unit.

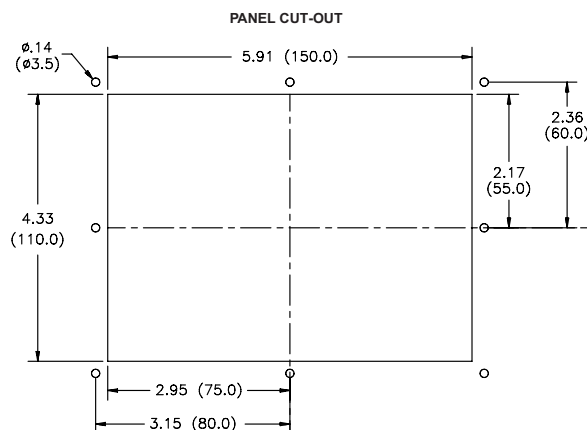
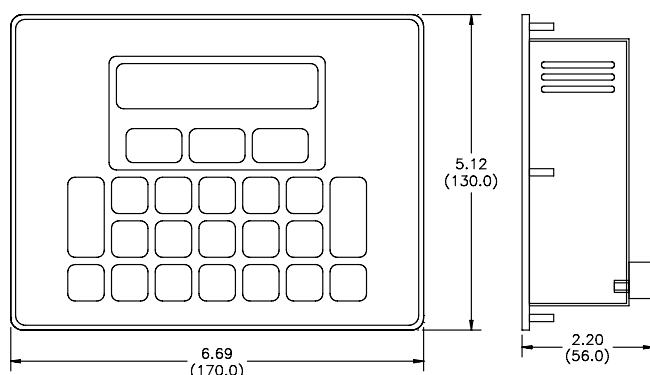
### ORDERING INFORMATION

MODEL NO.	DESCRIPTION	PART NO.
CL-05	LCD, 2 X 20, 3 Soft keys, 128 K memory	CL050000

### SPECIFICATIONS

- POWER REQUIREMENTS:** 11 min to 30 max. VDC @ 2.5 W  
**Power Up Current:** 2.5 A for 1 msec max.  
Must use a Class 2 or SELV rated power supply.
- DISPLAY:** 2 lines of 20 characters, 5 mm high liquid crystal display with bright LED backlight (with on/off software control)
- KEYPAD:** 3 screen legendable soft keys, numeric pad with raise, lower, next, previous, enter, delete, exit, alarms and mute keys.
- MEMORY:** 128 K (64 K user) battery backed RAM  
(Battery life expectancy 10 years)  
Optional factory fit expansion 256 K (192 K user)
- SERIAL PORTS:** One RS-232 for PC or printer connections, one RS232 and one RS485 for PLC connections up to 19200 Baud.
- PHYSICAL DIMENSIONS:** L = 170 mm, H = 130 mm, D = 56 mm.
- CONSTRUCTION:** Metal enclosure with NEMA 4/IP65 front plate when correctly fitted with the gasket provided. This unit is rated for NEMA 4/IP65 indoor use. Installation Category I, Pollution Degree 2.
- FIELD CONNECTIONS:** Removable screw terminal blocks.
- ENVIRONMENTAL CONDITIONS:**  
**Operating Temperature:** 0 to 40°C  
**Storage Temperature:** -20 to 80°C  
**Operating and Storage Humidity:** 80% max. relative humidity (non-condensing) from 0°C to 40°C.  
**Altitude:** Up to 2000 meters
- WEIGHT:** 1.6 lbs. (0.72 Kg)

### DIMENSIONS "In inches (mm)"



## MODEL CL10 - PARADIGM 2 X 20 LCD OPERATOR INTERFACE TERMINAL



- 2 LINE X 20 CHARACTER LIQUID CRYSTAL DISPLAY WITH LED BACKLIGHT
- 250 ALARM POINT LOGGER
- RECIPE HANDLING
- COMPREHENSIVE REPORT GENERATION
- UNLIMITED PASSWORD PROTECTION
- REAL TIME CLOCK, BATTERY BACKED
- EXPRESSION EVALUATION
- 32 BIT / FLOATING POINT MATH
- DIRECT, NETWORK ( One family of PLCs ) OR MODEM LINK TO PLC CONTROL SYSTEM
- NEMA 4/IP65 STEEL ENCLOSURE



### DESCRIPTION

The CL-10 from the Paradigm Range of operator interfaces meets the ever increasing demands of industry for powerful easy-to-use terminals. Both hardware and software are designed to allow the user to easily upgrade and take full advantage of our continuing development and improvements to our products.

### SAFETY SUMMARY

All safety related regulations, local codes and instructions that appear in the manual or on equipment must be observed to ensure personal safety and to prevent damage to either the instrument or equipment connected to it. If equipment is used in a manner not specified by the manufacturer, the protection provided by the equipment may be impaired.

Do not use this unit to directly command motors, valves, or other actuators not equipped with safeguards. To do so, can be potentially harmful to persons or equipment in the event of a fault to the unit.



**CAUTION:** Read complete instructions prior to installation and operation of the unit.

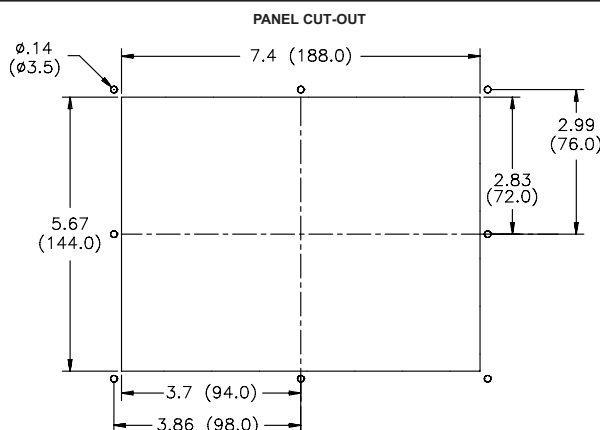
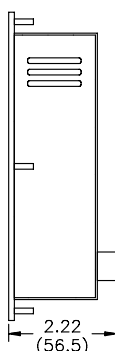
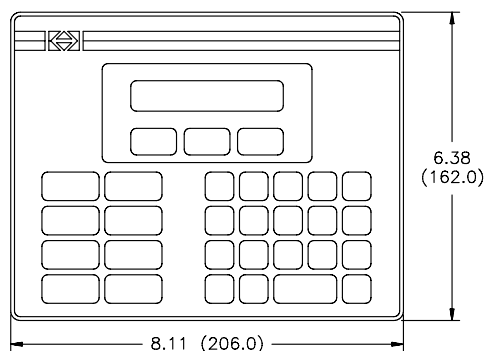
### ORDERING INFORMATION

MODEL NO.	DESCRIPTION	PART NO.
CL-10	LCD, 2 X 20, 8 Function, 3 Soft keys, 128 K memory	CL100000
	LCD, 2 X 20, 8 Function, 3 Soft keys, 256 K memory	CL100010

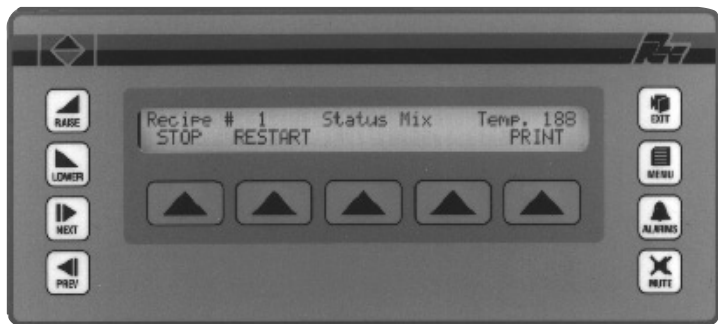
### SPECIFICATIONS

- POWER REQUIREMENTS:** 11 min to 30 max. VDC @ 2.5 W  
**Power Up Current:** 3.0A for 1 msec. max.  
Must use a Class 2 or SELV rated power supply.
- DISPLAY:** 2 lines of 20 characters, 5 mm high liquid crystal display with bright LED backlight (with on/off software control)
- KEYPAD:** 3 screen legendable soft keys, 8 User re-legendable function keys, numeric pad with raise, lower, next, previous, enter, delete, exit, alarms and mute keys, all with Tactile feedback.
- MEMORY:** 128 K (64 K user) battery backed RAM (Battery life expectancy 10 years). Optional factory fit expansion to 256 K (192 K user).
- SERIAL PORTS:** One RS-232 for PC or printer connections, one RS232 and one RS485 for PLC connection up to 19200 Baud.
- PHYSICAL DIMENSIONS:** L = 206 mm, H = 162 mm, D = 57 mm.
- CONSTRUCTION:** Metal enclosure with NEMA 4/IP65 front plate when correctly fitted with the gasket provided. This unit is rated for NEMA 4/IP65 indoor use. Installation Category I, Pollution Degree 2
- FIELD CONNECTIONS:** Removable screw terminal blocks.
- ENVIRONMENTAL CONDITIONS:**  
**Operating Temperature:** 0 to 40°C  
**Storage Temperature:** -20 to 80°C  
**Operating and Storage Humidity:** 80% max. relative humidity (non-condensing) from 0°C to 40°C.  
**Altitude:** Up to 2000 meters
- WEIGHT:** 2 lbs. (0.9 Kg)

### DIMENSIONS "In inches (mm)"



## MODEL CL15 - PARADIGM 2 X 40 LCD OPERATOR INTERFACE TERMINAL



- 2 LINE X 40 CHARACTER LIQUID CRYSTAL DISPLAY WITH LED BACKLIGHT
- 250 ALARM POINT LOGGER
- RECIPE HANDLING
- COMPREHENSIVE REPORT GENERATION
- UNLIMITED PASSWORD PROTECTION
- REAL TIME CLOCK, BATTERY BACKED
- EXPRESSION EVALUATION
- 32 BIT / FLOATING POINT MATH
- DIRECT, NETWORK ( One family of PLCs ) OR MODEM LINK TO PLC CONTROL SYSTEM
- NEMA 4/IP65 STEEL ENCLOSURE



### DESCRIPTION

The CL-15 from the Paradigm Range of operator interfaces meets the ever increasing demands of industry for powerful easy-to-use terminals. Both hardware and software are designed to allow the user to easily upgrade and take full advantage of our continuing development and improvements to our products.

### SAFETY SUMMARY

All safety related regulations, local codes and instructions that appear in the manual or on equipment must be observed to ensure personal safety and to prevent damage to either the instrument or equipment connected to it. If equipment is used in a manner not specified by the manufacturer, the protection provided by the equipment may be impaired.

Do not use this unit to directly command motors, valves, or other actuators not equipped with safeguards. To do so, can be potentially harmful to persons or equipment in the event of a fault to the unit.



**CAUTION:** Read complete instructions prior to installation and operation of the unit.

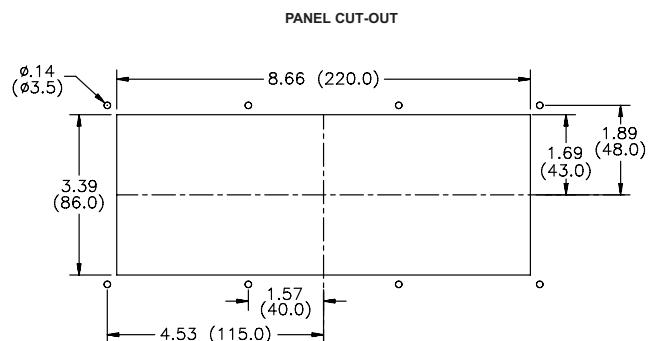
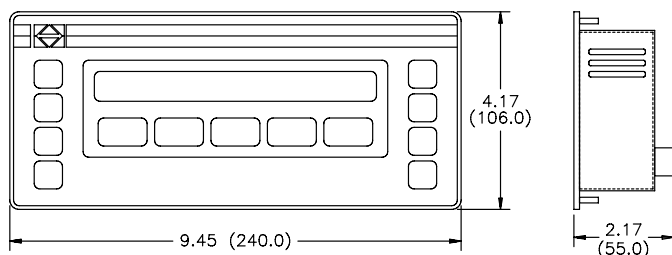
### ORDERING INFORMATION

MODEL NO.	DESCRIPTION	PART NO.
CL-15	LCD, 2 X 40, 5 Soft keys, 128 K memory	CL150000
	LCD, 2 X 40, 5 Soft keys, 256 K memory	CL150010

### SPECIFICATIONS

- POWER REQUIREMENTS:** 11 min. to 30 max. VDC @ 3.0 W  
**Power Up Current:** 2.5 A for 1 msec. max.  
 Must use a Class 2 or SELV rated power supply.
- DISPLAY:** 2 lines of 40 characters, 5 mm high liquid crystal display with bright LED backlight (with on/off software control)
- KEYPAD:** 5 screen legendable soft keys, raise, lower, next, previous, exit, menu, alarms and mute keys, all with Tactile feedback.
- MEMORY:** 128 K (64 K user) battery backed RAM (Battery life expectancy 10 years). Optional factory fit expansion to 256 K (192 K user).
- SERIAL PORTS:** One RS-232 for PC or printer connection, one RS232 and one RS485 for PLC connection up to 19200 Baud.
- PHYSICAL DIMENSIONS:** L = 240 mm, H = 160 mm, D = 55 mm.
- CONSTRUCTION:** Metal enclosure with NEMA 4/IP65 front plate when correctly fitted with the gasket provided. This unit is rated for NEMA 4/IP65 indoor use. Installation Category I, Pollution Degree 2
- FIELD CONNECTIONS:** Removable screw terminal blocks.
- ENVIRONMENTAL CONDITIONS:**  
**Operating Temperature:** 0 to 40°C  
**Storage Temperature:** -20 to 80°C  
**Operating and Storage Humidity:** 80% max. relative humidity (non-condensing) from 0°C to 40°C.  
**Altitude:** Up to 2000 meters
- WEIGHT:** 1.6 lbs. (0.72 Kg)

### DIMENSIONS "In inches (mm)"



# MODEL CL20 - PARADIGM 4 X 20 LCD OPERATOR INTERFACE TERMINAL



## GENERAL DESCRIPTION

The Model CL20 Operator Interface Terminal combines unique capabilities normally expected only from high-end units, at a very affordable price. The CL20 is configured using the same powerful EDICT97 Software as all Red Lion Paradigm Operator Interfaces. The result is savings in time to get challenging applications up and running, and frequent savings in hardware costs due to replacing many functions usually performed in separate expensive devices.

CL20 is robustly constructed for an industrial environment. With a metal enclosure and a non-corroding NEMA 4/IP65 front panel.

## SAFETY SUMMARY

All safety related regulations, local codes and instructions that appear in the manual or on equipment must be observed to ensure personal safety and to prevent damage to either the instrument or equipment connected to it. If equipment is used in a manner not specified by the manufacturer, the protection provided by the equipment may be impaired.

Do not use this unit to directly command motors, valves, or other actuators not equipped with safeguards. To do so, can be potentially harmful to persons or equipment in the event of a fault to the unit.



**CAUTION:** Read complete instructions prior to installation and operation of the unit.

## SPECIFICATIONS

- POWER REQUIREMENT:** 11 min. to 30 max. VDC @ 3.0 W  
**Power Up Current:** 3.0 A for 1msec. max.  
Must use a Class 2 or SELV rated power supply.
- DISPLAY:** 4 lines of 20 characters, 0.197" (5 mm) high liquid crystal display with bright LED backlight (with on/off software control)
- KEYPAD:** 3 screen legendable soft keys, 8 User re-legendable function keys, numeric pad with raise, lower, next, previous, enter, delete, exit, alarms and mute keys, all with Tactile feedback.
- MEMORY:** 128 K (64 K user) battery backed RAM (Battery life expectancy 10 years). Optional factory fit expansion to 256 K (192 K user).
- SERIAL PORTS:** Data Format and Baud Rates for each port is individually software programmable up to 19200 baud.  
Port 1: Programming Port - RS-232 on an RJ-11 jack.  
Port 2: RS-232 Port on a Plug-In Screw Terminal Block  
Port 3: RS-485 Port on a Plug-In Screw Terminal Block  
(Up to 20 units can be connected and individually addressed.)  
*Note: LED Indicators show communications status on Ports 2 & 3*

- 4 LINE X 20 CHARACTER LIQUID CRYSTAL DISPLAY WITH LED BACKLIGHT
- 250 ALARM POINT LOGGER
- RECIPE HANDLING
- COMPREHENSIVE REPORT GENERATION
- UNLIMITED PASSWORD PROTECTION
- REAL TIME CLOCK, BATTERY BACKED
- EXPRESSION EVALUATION
- 32 BIT / FLOATING POINT MATH
- DIRECT, NETWORK ( One family of PLCs ) OR MODEM LINK TO PLC CONTROL SYSTEM
- NEMA 4/IP65 STEEL ENCLOSURE



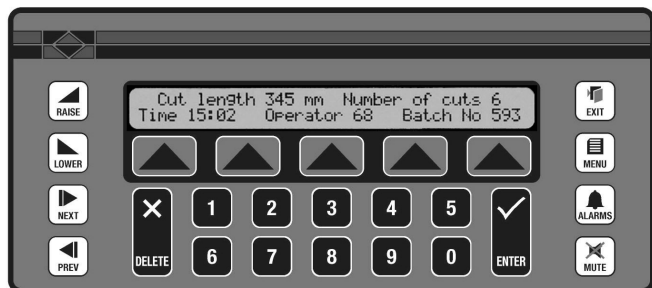
UL Recognized Component,  
File #E179259

- 6. COMMUNICATION MODES:** Any of the three ports can be used to communicate with Serial Devices. The CL20 may utilize one device protocol to operate as either a Master or Slave, on one port. If required CL20 may simultaneously communicate to Red Lion Serial devices, or a printer on a second port.
- 7. PHYSICAL DIMENSIONS:** L = 8.11" (206 mm), H = 6.38" (162 mm), D = 2.22" (56.5 mm).
- 8. CONSTRUCTION:** Metal enclosure with NEMA 4/IP65 front plate when correctly fitted with the gasket provided. This unit is rated for NEMA 4/IP65 indoor use. Installation Category I, Pollution Degree 2
- 9. FIELD CONNECTION:** Removable screw terminal blocks.
- 10. ENVIRONMENTAL CONDITIONS:**  
**Operating Temperature:** 0 to 40°C  
**Storage Temperature:** -20 to 80°C  
**Operating and Storage Humidity:** 80% max. relative humidity (non-condensing) from 0°C to 40°C.  
**Altitude:** Up to 2000 meters
- 11. CERTIFICATIONS AND COMPLIANCES:**  
UL Recognized Component, File #E179259  
Recognized to U.S. and Canadian requirements under the Component Recognition Program of Underwriters Laboratories, Inc  
**ELECTRICAL SAFETY**  
EN61010-1, IEC 1010-1  
Safety requirements for electrical equipment for measurement, control, and Laboratory use, Part 1  
**ELECTROMAGNETIC COMPATIBILITY**  
EN 50081-2 : 1994 Electromagnetic Compatibility Directive  
Generic Emission Standard  
Part 2 : Industrial Environment  
EN 50082-2 : 1994 Electromagnetic Compatibility Directive  
Generic Immunity Standard  
Part 2 : Industrial Environment  
EN 55022-B : 1995 Limits and Methods of Measurement of Radio Disturbance Characteristics of Information Technology Equipment
- 12. WEIGHT:** 2.1 lb. (0.95 Kg)

## ORDERING INFORMATION

MODEL NO.	DESCRIPTION	PART NUMBER
CL-20	LCD, 4 X 20, 8 Function, 3 Soft keys, 128 K memory	CL200000
	LCD, 4 X 20, 8 Function, 3 Soft keys, 256 K memory	CL200010

## MODEL CL40 - PARADIGM 2 X 40 LCD OPERATOR INTERFACE TERMINAL



### GENERAL DESCRIPTION

The Model CL40 Operator Interface Terminal combines unique capabilities normally expected only from high-end units, at a very affordable price. The CL40 is configured using the same powerful EDICT97 Software as all Red Lion Paradigm Operator Interfaces. The result is savings in time to get challenging applications up and running, and frequent savings in hardware costs due to replacing many functions usually performed in separate expensive devices.

CL40 is robustly constructed for an industrial environment. With a metal enclosure and a non-corroding NEMA 4/IP65 front panel.

### SAFETY SUMMARY

All safety related regulations, local codes and instructions that appear in the manual or on equipment must be observed to ensure personal safety and to prevent damage to either the instrument or equipment connected to it. If equipment is used in a manner not specified by the manufacturer, the protection provided by the equipment may be impaired.

Do not use this unit to directly command motors, valves, or other actuators not equipped with safeguards. To do so, can be potentially harmful to persons or equipment in the event of a fault to the unit.



**CAUTION:** Read complete instructions prior to installation and operation of the unit.

### ORDERING INFORMATION

MODEL NO.	DESCRIPTION	PART NUMBER
CL-40	LCD, 2 X 40, 5 Soft keys, 128 K memory	CL400000
	LCD, 2 X 40, 5 Soft keys, 256 K memory	CL400010

- 2 LINE X 40 CHARACTER LIQUID CRYSTAL DISPLAY WITH LED BACKLIGHT
- 250 ALARM POINT LOGGER
- RECIPE HANDLING
- COMPREHENSIVE REPORT GENERATION
- UNLIMITED PASSWORD PROTECTION
- REAL TIME CLOCK, BATTERY BACKED
- EXPRESSION EVALUATION
- 32 BIT / FLOATING POINT MATH
- DIRECT, NETWORK ( One family of PLCs ) OR MODEM LINK TO PLC CONTROL SYSTEM
- NEMA 4/IP65 STEEL ENCLOSURE

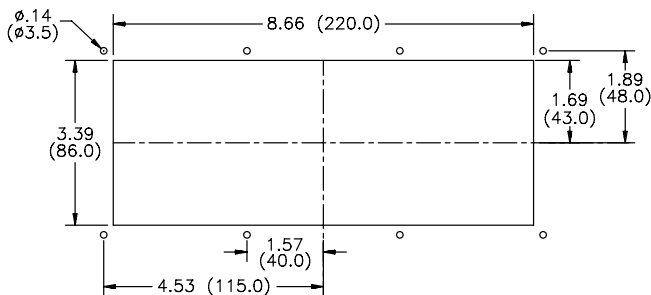
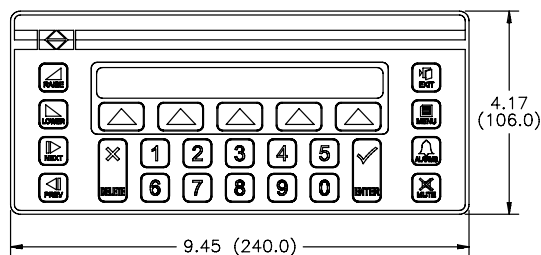


### SPECIFICATIONS

- POWER REQUIREMENT:** 11 min. to 30 max. VDC @ 3.0 W  
**Power Up Current:** 3.0 A for 1msec. max.  
Must use a Class 2 or SELV rated power supply.
- DISPLAY:** 2 lines of 40 characters, 0.197" (5 mm) high liquid crystal display with bright LED backlight (with on/off software control)
- KEYPAD:** 5 screen legendable soft keys, numeric pad with raise, lower, next, previous, enter, delete, exit, menu, alarms and mute keys, all with Tactile feedback.
- MEMORY:** 128 K (64 K user) battery backed RAM (Battery life expectancy 10 years). Optional factory fit expansion to 256 K (192 K user).
- SERIAL PORTS:** Data Format and Baud Rates for each port is individually software programmable up to 19200 baud.  
Port 1: Programming Port - RS-232 on an RJ-11 jack.  
Port 2: RS-232 Port on a Plug-In Screw Terminal Block  
Port 3: RS-485 Port on a Plug-In Screw Terminal Block  
(Up to 20 units can be connected and individually addressed.)  
*Note: LED Indicators show communications status on Ports 2 & 3*
- PHYSICAL DIMENSIONS:** L = 9.45" (240 mm), H = 6.3" (160 mm), D = 2.17" (55 mm).
- CONSTRUCTION:** Metal enclosure with NEMA 4/IP65 front plate when correctly fitted with the gasket provided. This unit is rated for NEMA 4/IP65 indoor use. Installation Category I, Pollution Degree 2
- FIELD CONNECTION:** Removable screw terminal blocks.
- ENVIRONMENTAL CONDITIONS:**  
**Operating Temperature:** 0 to 40°C  
**Storage Temperature:** -20 to 80°C  
**Operating and Storage Humidity:** 80% max. relative humidity (non-condensing) from 0°C to 40°C.  
**Altitude:** Up to 2000 meters
- WEIGHT:** 2.1 lb. (0.95 Kg)

### DIMENSIONS "In inches (mm)"

#### PANEL CUT-OUT



## MODEL CX100 - PARADIGM 2 X 20 VFD OPERATOR INTERFACE TERMINAL



- 2 LINE X 20 CHARACTER VACUUM FLUORESCENT DISPLAY
- 500 ALARM POINT LOGGER
- RECIPE HANDLING
- COMPREHENSIVE REPORT GENERATION
- UNLIMITED PASSWORD PROTECTION
- REAL TIME CLOCK, BATTERY BACKED
- EXPRESSION EVALUATION
- 32 BIT / FLOATING POINT MATH
- DIRECT, NETWORK (Including Multiple protocol) OR MODEM LINK TO PLC
- NEMA 4/IP65 STEEL ENCLOSURE



### DESCRIPTION

The CX-100 from the Paradigm Range of operator interfaces meets the ever increasing demands of industry for powerful easy-to-use terminals. Both hardware and software are designed to allow the user to easily upgrade and take full advantage of continuing development and improvements to our products.

### SAFETY SUMMARY

All safety related regulations, local codes and instructions that appear in the manual or on equipment must be observed to ensure personal safety and to prevent damage to either the instrument or equipment connected to it. If equipment is used in a manner not specified by the manufacturer, the protection provided by the equipment may be impaired.

Do not use this unit to directly command motors, valves, or other actuators not equipped with safeguards. To do so, can be potentially harmful to persons or equipment in the event of a fault to the unit.



**CAUTION:** Read complete instructions prior to installation and operation of the unit.

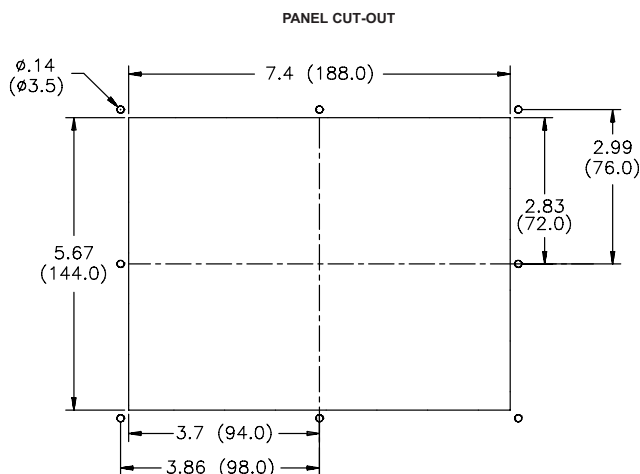
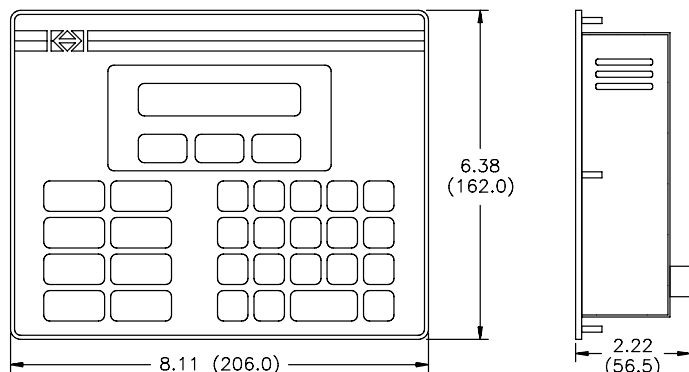
### ORDERING INFORMATION

MODEL NO.	DESCRIPTION	PART NO.
CX-100	VFD, 2 X 20, 8 Function keys, 128 K memory	CX100000
	VFD, 2 X 20, 8 Function keys, 256 K memory	CX100010

### SPECIFICATIONS

- POWER REQUIREMENTS:** 11 min. to 30 max. VDC @ 3.0 W  
**Power Up Current:** 2.75 A for 3.5 msec. max.  
Must use a Class 2 or SELV rated power supply.
- DISPLAY:** 2 lines of 20 characters, 5 mm high Vacuum Fluorescent display
- KEYPAD:** 3 screen legendable soft keys, 8 User re-legendable function keys, numeric pad with raise, lower, next, previous, enter, delete, exit, alarms and mute keys, all with Tactile feedback.
- MEMORY:** 128 K (64 K user) battery backed RAM (Battery life expectancy 10 years). Optional factory fit expansion to 256 K (192 K user).
- SERIAL PORTS:** One RS-232 for PC or printer connections, one RS232 and one RS485 for PLC connection up to 19200 Baud. (Can be used as a three port device for multiple protocol applications)
- PHYSICAL DIMENSIONS:** L = 206 mm, H = 162 mm, D = 57 mm.
- CONSTRUCTION:** Metal enclosure with NEMA 4/IP65 front plate when correctly fitted with the gasket provided. This unit is rated for NEMA 4/IP65 indoor use. Installation Category I, Pollution Degree 2
- FIELD CONNECTIONS:** Removable screw terminal blocks.
- ENVIRONMENTAL CONDITIONS:**  
**Operating Temperature:** 0 to 40°C  
**Storage Temperature:** -20 to 80°C  
**Operating and Storage Humidity:** 80% max. relative humidity (non-condensing) from 0°C to 40°C.  
**Altitude:** Up to 2000 meters
- WEIGHT:** 2.1 lbs. (0.95 Kg)

### DIMENSIONS "In inches (mm)"



## MODEL CX150 - PARADIGM 2 X 40 VFD OPERATOR INTERFACE TERMINAL



- 2 LINE X 40 CHARACTER VACUUM FLUORESCENT DISPLAY
- 500 ALARM POINT LOGGER
- RECIPE HANDLING
- COMPREHENSIVE REPORT GENERATION
- UNLIMITED PASSWORD PROTECTION
- REAL TIME CLOCK, BATTERY BACKED
- EXPRESSION EVALUATION
- 32 BIT / FLOATING POINT MATH
- DIRECT, NETWORK (Including Multiple protocols) OR MODEM LINK TO PLC
- NEMA 4/IP65 STEEL ENCLOSURE



### DESCRIPTION

The CX-150 from the Paradigm Range of operator interfaces meets the ever increasing demands of industry for powerful easy-to-use terminals. Both hardware and software are designed to allow the user to easily upgrade and take full advantage of our continuing development and improvements to our products.

### SAFETY SUMMARY

All safety related regulations, local codes and instructions that appear in the manual or on equipment must be observed to ensure personal safety and to prevent damage to either the instrument or equipment connected to it. If equipment is used in a manner not specified by the manufacturer, the protection provided by the equipment may be impaired.

Do not use this unit to directly command motors, valves, or other actuators not equipped with safeguards. To do so, can be potentially harmful to persons or equipment in the event of a fault to the unit.

### SPECIFICATIONS

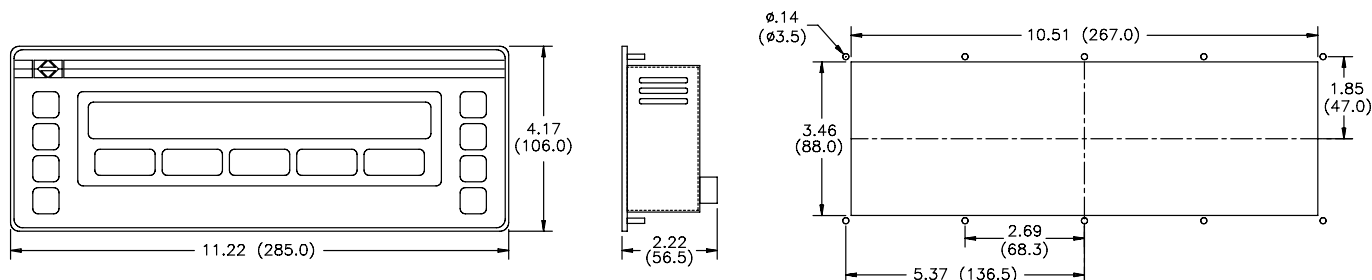
1. **POWER REQUIREMENTS:** 11 min. to 30 max. VDC @ 4.7 W  
**Power Up Current:** 2.5 A for 25 msec. max.  
 Must use a Class 2 or SELV rated power supply.
2. **DISPLAY:** 2 lines of 40 characters, 5 mm high Vacuum Fluorescent display
3. **KEYPAD:** 5 screen legendable soft keys, raise, lower, next, previous, exit, menu, alarms and mute keys, all with Tactile feedback.
4. **MEMORY:** 128 K (64 K user) battery backed RAM (Battery life expectancy 10 years). Optional factory fit expansion to 256 K (192 K user).
5. **SERIAL PORTS:** One RS-232 for PC or printer connection, one RS232 and one RS485 for PLC connection up to 19200 Baud. (Can be used as a three port device for multiple protocol applications)
6. **PHYSICAL DIMENSIONS:** L = 285 mm, H = 106 mm, D = 57 mm.
7. **CONSTRUCTION:** Metal enclosure with NEMA 4/IP65 front plate when correctly fitted with the gasket provided. This unit is rated for NEMA 4/IP65 indoor use. Installation Category I, Pollution Degree 2
8. **FIELD CONNECTIONS:** Removable screw terminal blocks.
9. **ENVIRONMENTAL CONDITIONS:**  
**Operating Temperature:** 0 to 40°C  
**Storage Temperature:** -20 to 80°C  
**Operating and Storage Humidity:** 80% max. relative humidity (non-condensing) from 0°C to 40°C.  
**Altitude:** Up to 2000 meters
10. **WEIGHT:** 2.1 lbs. (0.95 Kg)

### ORDERING INFORMATION

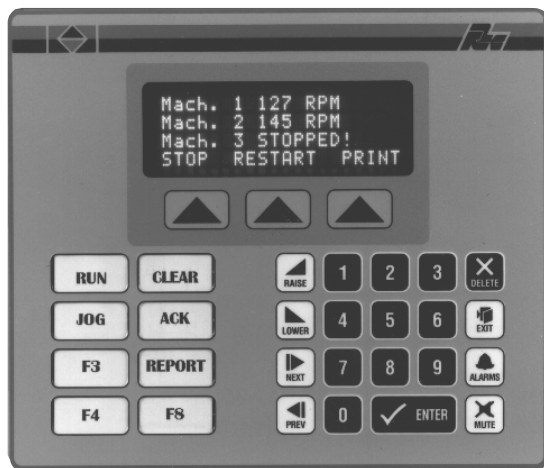
MODEL NO.	DESCRIPTION	PART NO.
CX-150	VFD, 4 X 20, 8 Function keys, 128 K memory	CX150000
	VFD, 4 X 20, 8 Function keys, 256 K memory	CX150010

### DIMENSIONS "In inches (mm)"

PANEL CUT-OUT



## MODEL CX200 - PARADIGM 4 X 20 VFD OPERATOR INTERFACE TERMINAL



- 4 LINE X 20 CHARACTER VACUUM FLUORESCENT DISPLAY
- 500 ALARM POINT LOGGER
- RECIPE HANDLING
- COMPREHENSIVE REPORT GENERATION
- UNLIMITED PASSWORD PROTECTION
- REAL TIME CLOCK, BATTERY BACKED
- EXPRESSION EVALUATION
- 32 BIT / FLOATING POINT MATH
- DIRECT, NETWORK (Including Multiple protocol) OR MODEM LINK TO PLC
- NEMA 4/IP65 STEEL ENCLOSURE



### DESCRIPTION

The CX-200 from the Paradigm Range of operator interfaces meets the ever increasing demands of industry for powerful easy-to-use terminals. Both hardware and software are designed to allow the user to easily upgrade and take full advantage of our continuing development and improvements to our products.

### SAFETY SUMMARY

All safety related regulations, local codes and instructions that appear in the manual or on equipment must be observed to ensure personal safety and to prevent damage to either the instrument or equipment connected to it. If equipment is used in a manner not specified by the manufacturer, the protection provided by the equipment may be impaired.

Do not use this unit to directly command motors, valves, or other actuators not equipped with safeguards. To do so, can be potentially harmful to persons or equipment in the event of a fault to the unit.



**CAUTION:** Read complete instructions prior to installation and operation of the unit.

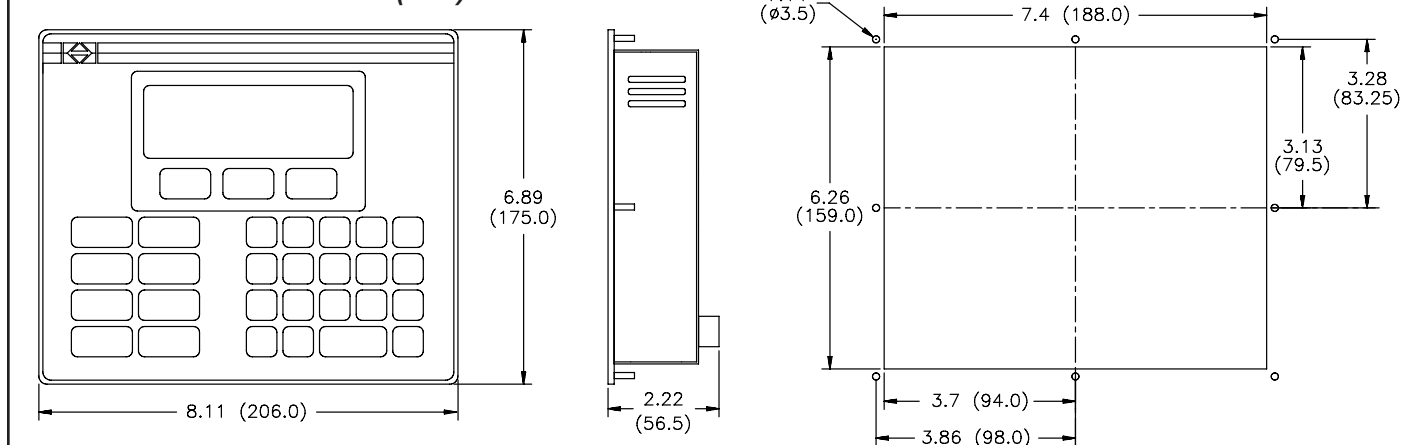
### ORDERING INFORMATION

MODEL NO.	DESCRIPTION	PART NO.
CX-200	VFD, 4 X 20, 8 Function keys, 128 K memory	CX200000
	VFD, 4 X 20, 8 Function keys, 256 K memory	CX200010

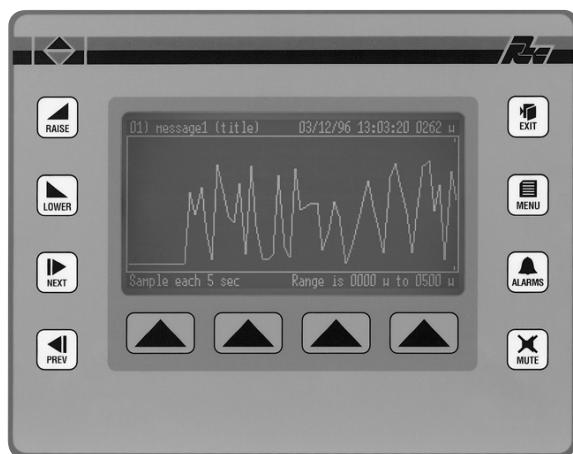
### SPECIFICATIONS

- POWER REQUIREMENTS:** 11 min to 30 max. VDC @ 3.2 W  
**Power Up Current:** 2.5 A for 7 msec. max.  
 Must use a Class 2 or SELV rated power supply.
- DISPLAY:** 4 lines of 20 characters, 5 mm high Vacuum Fluorescent display
- KEYPAD:** 3 screen legendable soft keys, 8 User re-legendable function keys, numeric pad with raise, lower, next, previous, enter, delete, exit, alarms and mute keys, all with Tactile feedback.
- MEMORY:** 128 K (64 K user) battery backed RAM (Battery life expectancy 10 years). Optional factory fit expansion to 256 K (192 K user).
- SERIAL PORTS:** One RS-232 for PC or printer connections, one RS232 and one RS485 for PLC connection up to 19200 Baud (Can be used as a three port device for multiple protocol applications)
- PHYSICAL DIMENSIONS:** L = 206 mm, H = 175 mm, D = 57 mm.
- CONSTRUCTION:** Metal enclosure with NEMA 4/IP65 front plate when correctly fitted with the gasket provided. This unit is rated for NEMA 4/IP65 indoor use. Installation Category I, Pollution Degree 2
- FIELD CONNECTIONS:** Removable screw terminal blocks.
- ENVIRONMENTAL CONDITIONS:**  
**Operating Temperature:** 0 to 40°C  
**Storage Temperature:** -20 to 80°C  
**Operating and Storage Humidity:** 80% max. relative humidity (non-condensing) from 0°C to 40°C.  
**Altitude:** Up to 2000 meters
- WEIGHT:** 2.3 lbs. (0.1 Kg)

### DIMENSIONS "In inches (mm)"



## MODEL GL300 - PARADIGM GRAPHICAL OPERATOR INTERFACE TERMINAL



- 256 X 128 PIXEL CCFL LIQUID CRYSTAL DISPLAY
- 500 ALARM POINT LOGGER
- COMPREHENSIVE REPORT GENERATION
- POWERFUL RECIPE HANDLING
- UNLIMITED PASSWORD PROTECTION
- REAL TIME CLOCK BATTERY BACKED
- EXPRESSION EVALUATION
- 32-BIT/FLOATING POINT MATHS
- DIRECT NETWORK (Including Multiple Protocol) OR MODEM LINK TO PLC
- NEMA 4/IP65 METAL ENCLOSURE



### DESCRIPTION

The Paradigm operator interface Model GL-300 was designed to meet the industrial demands of application power, versatility, reliability, and ease of use. The GL-300 has provision, common to all Paradigm Family products, allowing for future product upgrades as new options and capabilities are developed.

### SAFETY SUMMARY

All safety related regulations, local codes and instructions that appear in the manual or on equipment must be observed to ensure personal safety and to prevent damage to either the instrument or equipment connected to it. If equipment is used in a manner not specified by the manufacturer, the protection provided by the equipment may be impaired.

Do not use this unit to directly command motors, valves, or other actuators not equipped with safeguards. To do so, can be potentially harmful to persons or equipment in the event of a fault to the unit.



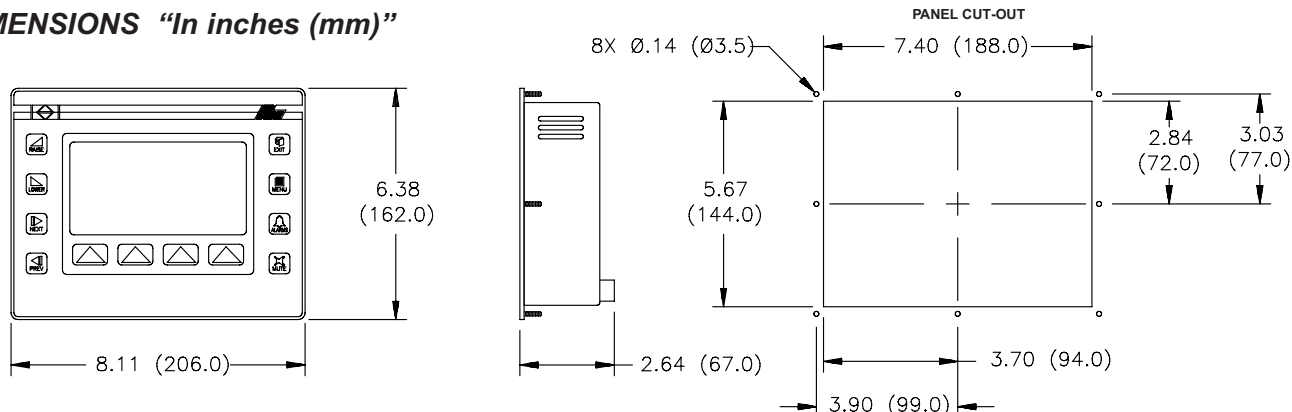
### ORDERING INFORMATION

MODEL NO.	DESCRIPTION	PART NUMBER
GL-300	256 X 128 CCFL, 16 X 40, 4 Soft keys, 256 K memory	GL300000
	256 X 128 CCFL, 16 X 40, 4 Soft keys, 768 K memory	GL300010

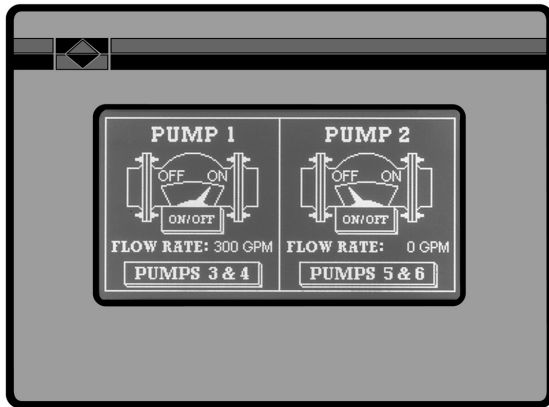
### SPECIFICATIONS

- POWER REQUIREMENTS:** 11 min. to 30 max. VDC @ 4.8 W  
**Power Up Current:** 2.5 A for 1 msec max.  
 Must use a Class 2 or SELV rated power supply.
- DISPLAY:** 256 x 128 pixel full graphic display with cold cathode backlight. Automatic temperature compensation. Text formats up to 16 x 40 characters.
- KEYPAD:** 4 screen legendable soft keys, raise, lower, next, previous, exit, menu, alarms and mute keys are all embossed and have tactile feedback.
- MEMORY:** 256K (192k user) battery backed RAM (Battery life expectancy 3 years 50/50 on/off cycle). Optional expansion to 768K (704K user).
- SERIAL PORTS:** One RS-232 for PC or printer connections.  
 One RS232 and one RS485 for PLC connections up to 19200 Baud. (Can be used as a three port device for multiple protocol applications.)
- ENVIRONMENTAL CONDITIONS:**  
**Operating Temperature:** 0 to 40°C  
**Storage Temperature:** -20 to 80°C  
**Operating and Storage Humidity:** 20 to 80% max. RH (non-condensing) from 0°C to 40°C.  
**Altitude:** Up to 2000 meters
- PHYSICAL DIMENSIONS:** L = 8.11" (206 mm), H = 6.38" (162 mm), D = 2.64" (67 mm).
- CONSTRUCTION:** Metal enclosure with NEMA 4/IP65 front plate when correctly fitted with the gasket provided. This unit is rated for NEMA 4/IP65 indoor use. Installation Category I, Pollution Degree 2
- FIELD CONNECTIONS:** Removable screw terminal blocks.
- WEIGHT:** 2.80 lb. (1.27 Kg.)

### DIMENSIONS "In inches (mm)"



# MODEL GL300T - MONOCHROME TOUCHSCREEN OPERATOR INTERFACE



- 256 X 128 PIXEL CCFL LIQUID CRYSTAL DISPLAY
- MULTIPLE LANGUAGE SUPPORT (UP TO 8 LANGUAGES)
- 500 ALARM POINT LOGGER
- COMPREHENSIVE REPORT GENERATION
- POWERFUL RECIPE HANDLING
- UNLIMITED PASSWORD PROTECTION
- REAL TIME CLOCK BATTERY BACKED
- EXPRESSION EVALUATION
- 32-BIT/FLOATING POINT MATH
- DIRECT NETWORK (Including Multiple Protocol) OR MODEM LINK TO PLC
- INDUSTRIAL TOUCHSCREEN
- NEMA 4/IP65 ALUMINUM ENCLOSURE



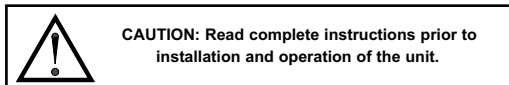
## DESCRIPTION

Model GL300T Operator Terminal combines unique capabilities normally expected only from expensive SCADA packages, with dramatic ease of use. The GL300T is configured using the same powerful EDICT 97 Software as all Red Lion Paradigm Operator Interfaces. The results are savings in time to get challenging applications up and running, and frequent savings in hardware costs due to replacing many functions usually performed in separate expensive devices.

## SAFETY SUMMARY

All safety related regulations, local codes and instructions that appear in the manual or on equipment must be observed to ensure personal safety and to prevent damage to either the instrument or equipment connected to it. If equipment is used in a manner not specified by the manufacturer, the protection provided by the equipment may be impaired.

Do not use this unit to directly command motors, valves, or other actuators not equipped with safeguards. To do so can be potentially harmful to persons or equipment in the event of a fault to the unit.



## GENERAL SPECIFICATIONS

- POWER REQUIREMENTS:** 11 VDC min. to 30 VDC max. @ 5.25 W  
Power Up Current: 2.5 A for 1 msec max.  
Must use a Class 2 or SELV rated power supply.
- DISPLAY:** 256 x 128 pixel full graphic display with cold cathode backlight. Automatic temperature compensation. Text formats up to 16 x 40 characters.
- MEMORY:** 768K (704K user) battery backed RAM (Battery life expectancy 3 years 50/50 on/off cycle).
- TOUCHSCREEN:** Continuous resistive touch screen interface specified for up to 5 million operations. 200 X 200 touch cells
- ENVIRONMENTAL CONDITIONS:**  
Operating Temperature: 0 to 50°C  
Storage Temperature: -20 to 60°C  
Operating and Storage Humidity: 20 to 80% max. RH (non-condensing) from 0°C to 50°C.  
Altitude: Up to 2000 meters
- CERTIFICATIONS AND COMPLIANCES:**  
**ELECTRICAL SAFETY**  
EN61010-1, IEC 1010-1  
Safety requirements for electrical equipment for measurement, control, and Laboratory use, Part 1

## ELECTROMAGNETIC COMPATIBILITY

- EN 50081-2 : 1994 Electromagnetic Compatibility Directive Generic Emission Standard; Part 2 : Industrial Environment  
EN 50082-2 : 1994 Electromagnetic Compatibility Directive Generic Immunity Standard; Part 2 : Industrial Environment  
EN 55022-B : 1995 Limits and Methods of Measurement of Radio Disturbance Characteristics of Information Technology Equipment

- PHYSICAL DIMENSIONS:** L = 7.65" (194.3 mm), H = 5.68" (144.3 mm), D = 2.10" (53.3 mm).
- CONSTRUCTION:** Metal enclosure with NEMA 4/IP65 front plate when correctly fitted with the gasket provided. This unit is rated for NEMA 4/IP65 indoor use. Installation Category I, Pollution Degree 2
- FIELD CONNECTIONS:** Removable screw terminal blocks.
- WEIGHT:** 2.32 lb. (1.05 Kg.)

## INPUT/OUTPUT COMMUNICATIONS SPECS

- SERIAL PORTS:** Data Format and Baud Rates for each port are individually software programmable up to 19200 baud.  
Port 1: Programming Port - RS-232 on an RJ-11 jack.  
Port 2: RS-232 Port on a Plug-In Screw Terminal Block  
Port 3: RS-485 Port on a Plug-In Screw Terminal Block  
(Up to 20 units can be connected and individually addressed.)  
*Note: LED Indicators show communications status on Ports 2 & 3*
- COMMUNICATION MODES:** Any of the three ports can be used to communicate with Serial Devices.  
Model - GL300T may communicate in Master mode with a different device protocol on each port (See Note).  
Ports 2 and 3 may be configured as different device protocols in Master mode and Port 1 may be used simultaneously in Slave mode for a third device protocol.  
However, only one of Ports 2 and 3 may be configured, if either is selected as a Slave protocol.  
*Note: Except if Allen Bradley DH485 is selected on either Port 2 or 3, in which case only Port 1 will be available for a separate Device Protocol.*

## ORDERING INFORMATION

MODEL NO.	DESCRIPTION	PART NUMBER
GL300T	256 X 128 CCFL, w/touchscreen, 768 K memory	GL300T00

# MODEL GL350 - PARADIGM GRAPHICAL OPERATOR INTERFACE TERMINAL



- MONOCHROME 256 x 128 PIXEL CCFL LIQUID CRYSTAL DISPLAY
- 500 ALARM POINT LOGGER
- COMPREHENSIVE REPORT GENERATION
- UNLIMITED PASSWORD PROTECTION
- REAL TIME CLOCK BATTERY BACKED
- EXPRESSION EVALUATION
- 32 BIT / FLOATING POINT MATH
- DIRECT, NETWORK (Including Multiple Protocol) OR MODEM LINK TO PLC
- NEMA 4/IP65 METAL ENCLOSURE



## DESCRIPTION

The Paradigm operator interface Model GL-350 was designed to meet the industrial demands of application power, versatility, reliability, and ease of use. The GL-350 has provision, common to all Paradigm Family products, allowing for future product upgrades and new options and capabilities are developed.

## SAFETY SUMMARY

All safety related regulations, local codes and instructions that appear in the manual or on equipment must be observed to ensure personal safety and to prevent damage to either the instrument or equipment connected to it. If equipment is used in a manner not specified by the manufacturer, the protection provided by the equipment may be impaired.

Do not use this unit to directly command motors, valves, or other actuators not equipped with safeguards. To do so, can be potentially harmful to persons or equipment in the event of a fault to the unit.

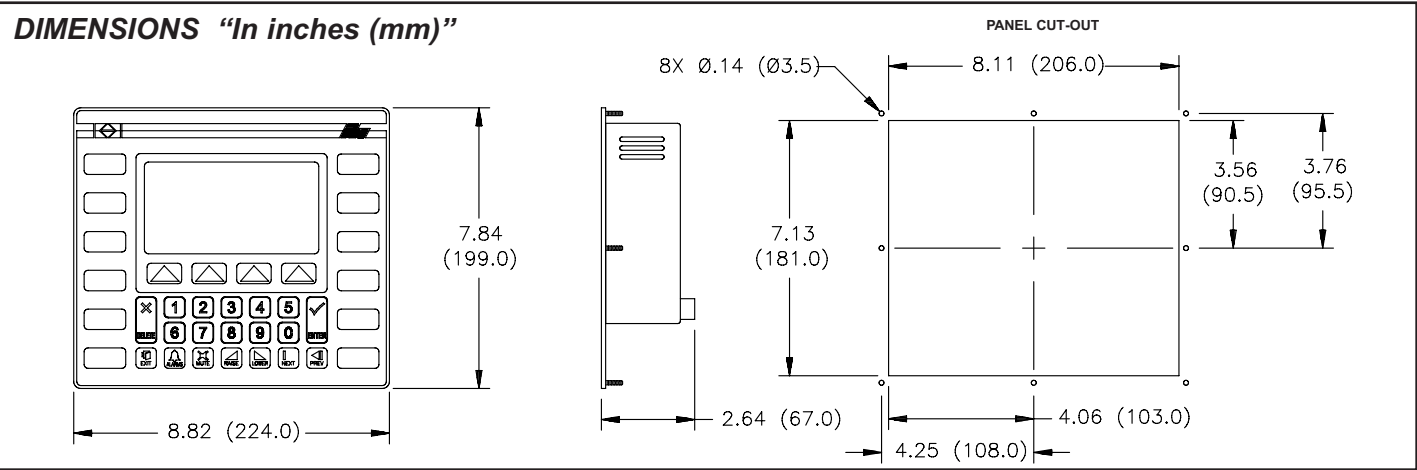


## ORDERING INFORMATION

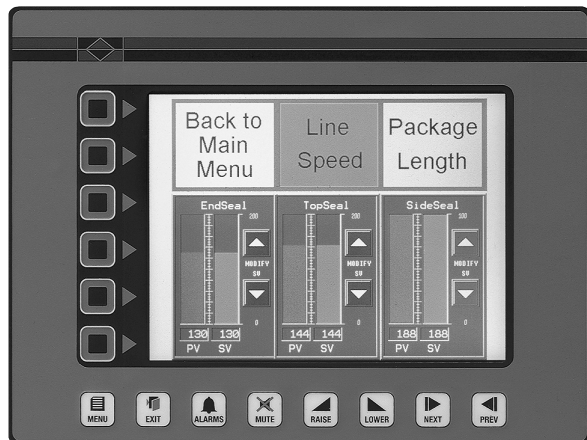
MODEL NO.	DESCRIPTION	PART NUMBER
GL-350	CCFL, 16 X 40, 12 Function, 4 Soft keys, 256 K memory	GL350000
	CCFL, 16 X 40, 12 Function, 4 Soft keys, 768 K memory	GL350010

## SPECIFICATIONS

- POWER REQUIREMENTS:** 11 min. to 30 max. VDC @ 4.8 W  
**Power Up Current:** 2.5 A for 1 msec. max.  
Must use a Class 2 or SELV rated power supply.
- DISPLAY:** 256 x 128 pixel monochrome display with cold cathode backlight. Automatic temperature compensation. Text formats upto 16 x 40 characters.
- KEYPAD:** 4 screen legendable soft keys, 12 User user-legendable function keys, numeric pad and raise, lower, next, previous, enter, delete, exit, alarms and mute keys all are embossed and have tactile feedback
- MEMORY:** 256K (192K user) battery backed RAM (Battery life expectancy 3 years 50/50 on/off cycle). Optional expansion to 768K (704K user).
- SERIAL PORTS:** One RS-232 for PC or printer connections. One RS232 and one RS485 for PLC connections up to 19200 Baud. (Can be used as a three port device for multiple protocol applications.)
- ENVIRONMENTAL CONDITIONS:**
  - Operating Temperature:** 0 to 40°C
  - Storage Temperature:** -20 to 80°C
  - Operating and Storage Humidity:** 20 to 80% max. RH (non-condensing) from 0°C to 40°C.
  - Altitude:** Up to 2000 meters
- PHYSICAL DIMENSIONS:** L = 8.82" (224 mm), H = 7.84" (199 mm), D = 2.64" (67 mm).
- CONSTRUCTION:** Metal enclosure with NEMA 4/IP65 front plate when correctly fitted with the gasket provided. This unit is rated for NEMA 4/IP65 indoor use. Installation Category I, Pollution Degree 2
- FIELD CONNECTIONS:** Removable screw terminal blocks.
- WEIGHT:** 3.04 lb. (1.38 Kg.)



## MODELS VX-500 & VX-500T - COLOR GRAPHICAL & COLOR GRAPHICAL WITH TOUCHSCREEN OPERATOR TERMINALS



- NEMA 4/IP65 ALUMINUM ENCLOSURE
- DIRECT NETWORK (Including Multiple Protocol) OR MODEM LINK TO PLC

- 640 x 480 PIXEL CCFL LIQUID CRYSTAL DISPLAY-7.5" DIAGONAL DSTN COLOR-FULL VGA (16 colors)
- POWERFUL 32-BIT PROCESSOR AND ACCELERATED GRAPHICS CONTROLLER FOR HIGH PERFORMANCE LEVELS
- MULTIPLE LANGUAGE SUPPORT (UP TO 8 LANGUAGES)
- ANIMATED PowerPoint® STYLE PAGE TRANSITIONS
- SLIDEOUT SOFTKEY MENUS
- 500 ALARM POINT LOGGER
- COMPREHENSIVE REPORT GENERATION
- POWERFUL RECIPE HANDLING
- UNLIMITED PASSWORD PROTECTION
- REAL TIME CLOCK BATTERY BACKED
- FORM C RELAY OUTPUT
- EXPRESSION EVALUATION
- 32-BIT/FLOATING POINT MATH



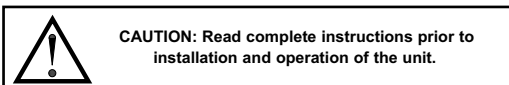
### DESCRIPTION

The VX-500 and VX-500T from the Paradigm Range of operator interfaces meet the ever increasing demands of industry for powerful easy-to-use terminals. Both hardware and software are designed to allow the user to easily upgrade and take full advantage of our continuing development and improvements to our products.

### SAFETY SUMMARY

All safety related regulations, local codes and instructions that appear in the manual or on equipment must be observed to ensure personal safety and to prevent damage to either the instrument or equipment connected to it. If equipment is used in a manner not specified by the manufacturer, the protection provided by the equipment may be impaired.

Do not use this unit to directly command motors, valves, or other actuators not equipped with safeguards. To do so can be potentially harmful to persons or equipment in the event of a fault to the unit.



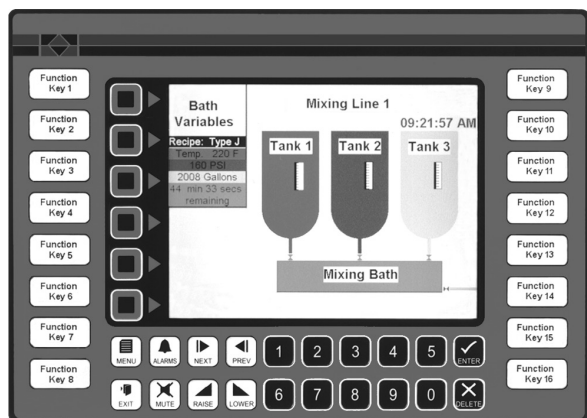
### ORDERING INFORMATION

MODEL NO.	DESCRIPTION	PART NUMBER
VX-500	640 X 480 CCFL, Full VGA Color, 40 X 30, 6 Soft keys, 736 K memory (672 K user)	VX500000
	640 X 480 CCFL, Full VGA Color, 40 X 30, 6 Soft keys, 736 K memory (672 K user). Enhanced Display Version	VX500P00
VX-500T	640 X 480 CCFL, Full VGA Color, 40 X 30, 6 Soft keys, W/Touchscreen 736 K memory (672 K user)	VX500T00
	640 X 480 CCFL, Full VGA Color, 40 X 30, 6 Soft keys, W/Touchscreen 736 K memory (672 K user). Enhanced Display Version	VX500TP0

### SPECIFICATIONS

- POWER REQUIREMENTS:** 15 VDC min. to 30 VDC max. @ 9.75 W  
**Power Up Current:** 2.5 A for 4 msec max.  
Must use a Class 2 or SELV rated power supply.
- DISPLAY:** 640 x 480 pixels (7.5 inch diagonal) CCFL Liquid Crystal DSTN color full VGA display. Text formats up to 40 x 30 characters. (VX500P00 and VX500TP0 optional enhanced display with 50% increased brightness and significantly improved viewing angles.)
- KEYPAD:** 6 screen legendable soft keys, raise, lower, next, previous, exit, menu, alarms and mute keys are all embossed and have tactile feedback.
- TOUCHSCREEN (VX500T only):** Continuous resistive touch screen interface specified for up to 5 million operations. 200 X 200 touch cells
- MEMORY:** 736K (672K user) battery backed RAM (Battery life expectancy 3 years 50/50 on/off cycle).
- SERIAL PORTS:** One RS-232 for PC or printer connections.  
One RS232 and one RS485 for PLC connections up to 19200 Baud. (Can be used as a three port device for multiple protocol applications.)
- RELAY OUTPUT:** Form C relay output 1/2A @ 125 VAC, 1 A @24 VDC
- ENVIRONMENTAL CONDITIONS:**  
**Operating Temperature:** 0 to 50°C  
**Storage Temperature:** -20 to 80°C  
**Operating and Storage Humidity:** 20 to 80% max. RH (non-condensing) from 0°C to 40°C.  
**Altitude:** Up to 2000 meters
- PHYSICAL DIMENSIONS:** L = 9.77" (248.2 mm), H = 7.37" (187.2 mm), D = 2.1" (53.3 mm).
- CONSTRUCTION:** Metal enclosure with NEMA 4/IP65 front plate when correctly fitted with the gasket provided. This unit is rated for NEMA 4/IP65 indoor use. Installation Category I, Pollution Degree 2
- FIELD CONNECTIONS:** Removable screw terminal blocks.
- WEIGHT:** 2.94 lb. (1.33 Kg.)

## MODEL VX-550 - PARADIGM COLOR GRAPHICAL OPERATOR TERMINALS



- DIRECT NETWORK (Including Multiple Protocol) OR MODEM LINK TO PLC
- 16 USER LEGENDABLE FUNCTION KEYS

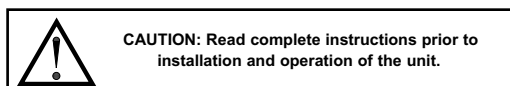
### DESCRIPTION

The VX-550 from the Paradigm Range of operator interfaces meets the ever increasing demands of industry for powerful easy-to-use terminals. Both hardware and software are designed to allow the user to easily upgrade and take full advantage of our continuing development and improvements to our products.

### SAFETY SUMMARY

All safety related regulations, local codes and instructions that appear in the manual or on equipment must be observed to ensure personal safety and to prevent damage to either the instrument or equipment connected to it. If equipment is used in a manner not specified by the manufacturer, the protection provided by the equipment may be impaired.

Do not use this unit to directly command motors, valves, or other actuators not equipped with safeguards. To do so can be potentially harmful to persons or equipment in the event of a fault to the unit.



### SPECIFICATIONS

- POWER REQUIREMENTS:** 15 VDC min. to 30 VDC max. @ 9.75 W  
**Power Up Current:** 2.5 A for 4 msec max.  
Must use a Class 2 or SELV rated power supply.
- DISPLAY:** 640 x 480 pixels (7.75 inch diagonal) CCFL Liquid Crystal DSTN color full VGA display. Text formats up to 40 x 30 characters.
- KEYPAD:** 6 screen legendable soft keys, 16 User-legendable Function keys, raise, lower, next, previous, exit, menu, alarms and mute keys are all embossed and have tactile feedback.
- MEMORY:** 736K (672K user) battery backed RAM (Battery life expectancy 3 years 50/50 on/off cycle).
- SERIAL PORTS:** One RS-232 for PC or printer connections.  
One RS232 and one RS485 for PLC connections up to 19200 Baud. (Can be used as a three port device for multiple protocol applications.)
- RELAY OUTPUT:** Form C relay output 1/2A @ 125VAC, 1 A @ 24 VDC

- 640 x 480 PIXEL CCFL LIQUID CRYSTAL DISPLAY-7.75" DIAGONAL DSTN COLOR-FULL VGA (16 colors)
- POWERFUL 32-BIT PROCESSOR AND ACCELERATED GRAPHICS CONTROLLER FOR HIGH PERFORMANCE LEVELS
- MULTIPLE LANGUAGE SUPPORT (UP TO 8 LANGUAGES)
- ANIMATED PowerPoint® STYLE PAGE TRANSITIONS
- SLIDE OUT SOFTKEY MENUS
- 500 ALARM POINT LOGGER
- COMPREHENSIVE REPORT GENERATION
- POWERFUL RECIPE HANDLING
- UNLIMITED PASSWORD PROTECTION
- REAL TIME CLOCK BATTERY BACKED
- FORM C RELAY OUTPUT
- EXPRESSION EVALUATION
- 32-BIT/FLOATING POINT MATH
- NEMA 4/IP65 ALUMINUM ENCLOSURE



### 7. CERTIFICATIONS AND COMPLIANCES:

#### ELECTRICAL SAFETY

EN61010-1, IEC 1010-1

Safety requirements for electrical equipment for measurement, control, and Laboratory use, Part 1

#### ELECTROMAGNETIC COMPATIBILITY

EN 50081-2 : 1994 Electromagnetic Compatibility Directive Generic Emission Standard  
Part 2 : Industrial Environment

EN 50082-2 : 1994 Electromagnetic Compatibility Directive Generic Immunity Standard  
Part 2 : Industrial Environment

EN 55022-B : 1995 Limits and Methods of Measurement of Radio Disturbance Characteristics of Information Technology Equipment

### 8. ENVIRONMENTAL CONDITIONS:

**Operating Temperature:** 0 to 50°C

**Storage Temperature:** -20 to 80°C

**Operating and Storage Humidity:** 20 to 80% max. RH (non-condensing) from 0°C to 40°C.

**Altitude:** Up to 2000 meters

9. **PHYSICAL DIMENSIONS:** L = 11.27" (286.3 mm), H = 7.94" (201.7 mm), D = 2.1" (53.3 mm).

10. **CONSTRUCTION:** Metal enclosure with NEMA 4/IP65 front plate when correctly fitted with the gasket provided. This unit is rated for NEMA 4/IP65 indoor use. Installation Category I, Pollution Degree 2

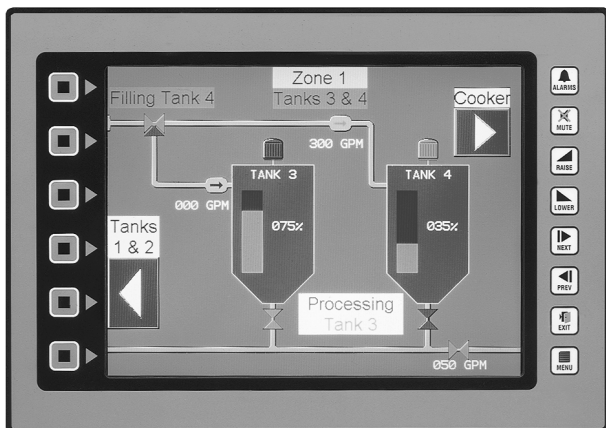
11. **FIELD CONNECTIONS:** Removable screw terminal blocks.

12. **WEIGHT:** 3.20 lb. (1.45 Kg.)

### ORDERING INFORMATION

MODEL NO.	DESCRIPTION	PART NUMBER
VX-550	640 X 480 CCFL, Full VGA Color, 40 X 30, 16 Legendable Function keys, 6 Soft keys, 736 K memory (672 K user)	VX550S00

## MODEL TX700T - COLOR TFT TOUCHSCREEN OPERATOR INTERFACE



- NEMA 4/IP65 ALUMINUM ENCLOSURE
- DIRECT NETWORK (Including Multiple Protocol) OR MODEM LINK TO PLC

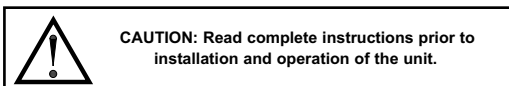
### DESCRIPTION

Model TX700T Operator Terminal combines unique capabilities normally expected only from expensive SCADA packages, with dramatic ease of use. TX700T is configured using the same powerful EDICT97 Software as all Red Lion Paradigm Operator Interfaces. The results are savings in time to get challenging applications up and running, and frequent savings in hardware costs due to replacing many functions usually performed in separate expensive devices.

### SAFETY SUMMARY

All safety related regulations, local codes and instructions that appear in the manual or on equipment must be observed to ensure personal safety and to prevent damage to either the instrument or equipment connected to it. If equipment is used in a manner not specified by the manufacturer, the protection provided by the equipment may be impaired.

Do not use this unit to directly command motors, valves, or other actuators not equipped with safeguards. To do so can be potentially harmful to persons or equipment in the event of a fault to the unit.



### GENERAL SPECIFICATIONS

- POWER REQUIREMENTS:** 15 VDC min. to 30 VDC max. @ 15.25 W  
Power Up Current: 2.5 A for 4 msec max.  
Must use a Class 2 or SELV rated power supply.
- DISPLAY:** 640 x 480 pixels (10.4") CCFL Liquid Crystal TFT color full VGA display. Text formats up to 40 x 30 characters.
- KEYPAD:** 6 screen legendable soft keys, raise, lower, next, previous, exit, menu, alarms and mute keys are all embossed and have tactile feedback.
- TOUCHSCREEN:** Continuous resistive touch screen interface specified for up to 5 million operations. 200 X 200 touch cells.
- MEMORY:** 736K (672k user) battery backed RAM (Battery life expectancy 3 years 50/50 on/off cycle).
- RELAY OUTPUT:** Form C relay output 1/2A @ 125 VAC, 1 A @24 VDC
- ENVIRONMENTAL CONDITIONS:**  
Operating Temperature: 0 to 50°C  
Storage Temperature: -20 to 60°C  
Operating and Storage Humidity: 20 to 80% max. RH (non-condensing) from 0°C to 50°C.  
Altitude: Up to 2000 meters

- 10.4" BRIGHT, WIDE VIEWING ANGLE VGA COLOR DISPLAY  
640 X 480 PIXELS CCFL TFT (16 colors)
- 200 X 200 CELL RESISTIVE TOUCHSCREEN
- POWERFUL 32-BIT PROCESSOR AND ACCELERATED GRAPHICS CONTROLLER FOR HIGH PERFORMANCE LEVELS
- MULTIPLE LANGUAGE SUPPORT (UP TO 8 LANGUAGES)
- ANIMATED PowerPoint® STYLE PAGE TRANSITIONS
- SLIDEOUT SOFTKEY MENUS
- 500 ALARM POINT LOGGER
- COMPREHENSIVE REPORT GENERATION
- POWERFUL RECIPE HANDLING
- UNLIMITED PASSWORD PROTECTION
- REAL TIME CLOCK BATTERY BACKED
- FORM C RELAY OUTPUT
- EXPRESSION EVALUATION
- 32-BIT/FLOATING POINT MATH



### 8. CERTIFICATIONS AND COMPLIANCES:

#### ELECTRICAL SAFETY

EN61010-1, IEC 1010-1

Safety requirements for electrical equipment for measurement, control, and Laboratory use, Part 1

#### ELECTROMAGNETIC COMPATIBILITY

- EN 50081-2 : 1994 Electromagnetic Compatibility Directive Generic Emission Standard; Part 2 : Industrial Environment
- EN 50082-2 : 1994 Electromagnetic Compatibility Directive Generic Immunity Standard; Part 2 : Industrial Environment
- EN 55022-B : 1995 Limits and Methods of Measurement of Radio Disturbance Characteristics of Information Technology Equipment

9. **PHYSICAL DIMENSIONS:** L = 12.16" (308.5 mm), H = 8.59" (218 mm), D = 2.14" (66.8 mm).

10. **CONSTRUCTION:** Metal enclosure with NEMA 4/IP65 front plate when correctly fitted with the gasket provided. This unit is rated for NEMA 4/IP65 indoor use. Installation Category I, Pollution Degree 2

11. **FIELD CONNECTIONS:** Removable screw terminal blocks.

12. **WEIGHT:** 6.20 lb. (2.81 Kg.)

### INPUT/OUTPUT COMMUNICATIONS SPECS

- SERIAL PORTS:** Data Format and Baud Rates for each port are individually software programmable up to 19200 baud.  
Port 1: Programming Port - RS-232 on an RJ-11 jack.  
Port 2: RS-232 Port on a Plug-In Screw Terminal Block  
Port 3: RS-485 Port on a Plug-In Screw Terminal Block  
(Up to 32 units can be connected and individually addressed.)  
*Note: LED Indicators show communications status on Ports 2 & 3*
- COMMUNICATION MODES:** Any of the three ports can be used to communicate with Serial Devices.  
Model - TX700T may communicate in Master mode with a different device protocol on each port (See Note).  
Ports 2 and 3 may be configured as different device protocols in Master mode and Port 1 may be used simultaneously in Slave mode for a third device protocol. However, only one of Ports 2 and 3 may be configured, if either is selected as a Slave protocol.  
*Note: Except if Allen Bradley DH485 is selected on either Port 2 or 3, in which case only Port 1 will be available for a separate Device Protocol.*

### ORDERING INFORMATION

MODEL NO.	DESCRIPTION	PART NUMBER
TX700T	640 X 480 CCFL, Full VGA Color TFT, 40 X 30, W/Touchscreen 736 K memory	TX700T00

# COMMON FEATURES FOR GRAPHIC BASED OPERATOR TERMINALS

## PROGRAMMABILITY

### Event Driven Configuration Tool

EDICT 97, an extremely powerful Windows 95/3.11 based software program, provides for the intuitive configuration of every aspect of the operator interface's behavior. The requirement for time consuming PLC ladder logic is drastically reduced by the unique event driven approach of EDICT 97. The capability of this program, in conjunction with the PLC and the Paradigm operator interface unit, ensures a great deal of advanced functionality for your system. This powerful PLC/Paradigm system provides many of the capabilities and features normally associated with the more complicated and costly PC/SCADA systems. Display pages are easily generated, including PLC and internal variables, text strings, or bar charts. All dynamic elements are also available as alarms, recipes, triggers, and reports for the run time software. After completion of the programming, the program is directly downloaded to the operator interface from your PC, without any compiling or saving requirement. When you require a change in your program, EDICT 97 loads only the change, not the entire program, saving valuable on-line time.

### DYNAMIC DISPLAY PAGE ELEMENTS

Each display page has provisions to show static and dynamic information, including data variables, text messages, time, and date.

**Data Variables** can be either PLC derived or internally generated, either in data entry or display only mode. The Paradigm unit has an extremely powerful math capability, allowing the operator to manipulate the variables to meet the specific application's demands. If required, the display can be formatted to BCD, binary, hex, floating point, and string. Upper and lower limits of data entry variables are fully supported and password protected.

**Text Message Animation** enables several different types of animated text from a local or global message table to be displayed. The message displayed is dependent on the condition of the particular controlling expression. The controlling expression may be a PLC bit level, a timer value, preset counter condition, or any one of a wide variety of message triggers.

**Time and Date** in the Paradigm unit has the capability to display in any combination of year, month, day, hours, minutes, and seconds.

**Bar Graphs** in horizontal format are easily attached to data variables. The partial or full length bar graph displays can be scaled and offset to optimize the required display effect.

## SECURITY

The password protection scheme provides the ultimate in tamper-proof capability. Access can be limited on a unit, page, recipe, or even individual data entries.

## ALARMS

The Paradigm unit can monitor and log up to 500 alarms. Such triggers as a simple bit level transition, a PLC coil activation, or a complex application algorithm can activate an alarm. The alarms can be time and date stamped, with an automatic screen display and/or downloading to a printer for hard copy recording purposes.

## RECIPE HANDLING

Recipe handling in the Paradigm Operator Interfaces can be tailored to your requirements. Using the "Data Files" section of Named Data, one can set up arrays with meaningful titles, and select, edit, and maintain, recipe data up to 8000 elements per file. In conjunction with User Programs, and the flexible data displays, the operator can select desired recipe, by number or by title, and either upload from, or download to, the target system. All the functions of EDICT97 are available, so the programmer can password protect the editing of the recipes and allow for the transfer of data from a host system.

## REAL TIME SCHEDULE

Real time schedule allows for repetitive or one time task to take place in the system. Typically a schedule action similar to...At 1:55 PM on Monday, Wednesday, and Friday print the production report...is required in the application. In conjunction with the recipe capabilities, a downloading of a special recipe can be requested by the real time schedule feature.

## MULTIPLE LANGUAGE SUPPORT

This powerful feature allows users to program the text in their databases in up to 8 different languages. A system variable entry makes it easy for end users to select one of the preprogrammed languages. EDICT-97 features powerful language editing tools for easy implementation.

## USER PROGRAMS

This feature offers the user the ability to incorporate custom application requirements via a powerful program language. For example, a program designated "Calculate Volume" which determines the amount of fluid in a round tank at specific temperatures could be created. This program would be triggered to run and display each time the page denoted as "Volume Now" is requested. The ability to customize to your applications specialized needs is easily solved with the user program capability.

## KEYBOARD EDITING

All the interface keys can be programmed to perform virtually unlimited functions with each key, having multiple actions assigned to three types of key events: key pressed, key held down (auto repeat), and key released. Typical key actions would be Gotopage, set value, load recipe, view alarms, print report, and many more.

## COMMUNICATIONS

With over 70 communication drivers available, the Paradigm operator interface offers a wide range of connectivity including: PLCs, Variable Speed Drives, Temperature Controllers, Bar Code Readers, etc. Utilizing real PLC data references, the automatic comms configuration optimizes the system's communication performance. In the event that your specific driver does not appear on the Paradigm drivers list, let us know, as this list is always being expanded to meet our customers' needs.

## GRAPHIC UNITS

In addition to all the features of the character-based units, the graphic units will provide exceptional value in displaying trend graphs, process schematics and flow, and others, limited only by the imagination of the designer. The programmer can use the built-in standard symbols, or construct them. A sequence of graphical symbols can be assigned to a PLC location, and the powerful software will step through the sequence without the necessity of programming multiple expressions for each bitmap. Some of the inherent features of the Graphical Display units:

- Data Logging
- Process symbols, such as tanks, valves, etc.
- Extraordinary color displays on the VX-500, VX-550 and TX700T.
- Memory expansion is field-upgradeable.
- Plus all the functions available in EDICT 97, the powerful event driven configuration tool that allows one to configure a system to do what is needed.

## ANIMATED GRAPHICS

Graphical pages are constructed using both bitmaps and object graphics. Animation items such as tending, tank filling, horizontal and vertical bar graphs, valves, etc., make your display pages aesthetically pleasing as well as informative to the operator.

## TOUCH-SCREEN

These units are fitted with a continuous resolution resistive touch-screen, providing an effective resolution of over 200 by 200 cells. This allows touch-sensitive objects to be placed anywhere on the screen, without restricting your designs to the coarse grid employed by competitive products. The touch-screen is fully operable with gloved hands and is specified for up to 5 million operations.

## HARDWARE INFORMATION

This bulletin contains a variety of information related to the installation and operation of the Operator Interface supplied. Ideally, you should read this document thoroughly before attempting to use the equipment. For information about the software aspects of the terminal, please consult other documentation.

## CONTENTS OF PACKAGE

The Operator Interface is supplied in a packaging box containing the following...

- ◆ The interface terminal itself.
- ◆ A NEMA 4/IP65 rated mounting gasket.
- ◆ A bag containing panel hardware.
- ◆ A Function Key Strip for units with Function Keys.
- ◆ This hardware bulletin.
- ◆ If any of these items is missing, please contact your supplier immediately.

## FUNCTION KEY STRIPS

The function keys have clear windows that permit the user to insert labels appropriate to the process. A formatted page is supplied upon which the user can enter function names (e.g. RUN, PRINT, etc.). These strips are inserted from the rear of the panel through slots below the function keys located underneath the gasket.

Take care that the ink applied will not rub off of the paper, or else blemishes will be left on the inside of the window. Laminated paper or plastic film can prove easier to insert than normal photocopier paper. It also helps if the starting edge of the paper has about 0.25 inches of its corners cut off at a 45 degree angle.

*Note: Add an additional 1.5" to label length to allow for easier insertion and removal.*

## POWER SUPPLY REQUIREMENTS

The Operator Interface requires a regulated 15 to 30 VDC power supply rated at 9.75 W unless otherwise stated on the label.

- ◆ The terminal may take as little as 300 mA in certain circumstances, so be sure that the chosen power supply can operate correctly with this load. Large switch-mode supplies tend to need a certain minimum load before they will operate correctly.

In any case, it is very important that the power supply is mounted correctly if the unit is to operate reliably. A very high proportion of reported problems are caused by incorrect power supply installation, so please take care to observe the following points...

- ◆ The power supply must be mounted close to the unit, with usually not more than 6 feet of cable between the supply and the Operator Interface. Ideally, as short a length as is possible should be used. In particular, the power supply should not be mounted on the back of the panel when the Operator Interface is installed in the panel door unless a short cable run can be achieved.
- ◆ The wire used to connect the Operator Interface's power supply should be of at least 22 gauge wire. If a longer cable run is used, you should use even thicker cable. The routing of the cable should be kept away from large contactors, inverters and other devices which may generate significant electrical noise.

## BATTERY BACKUP ISSUES

The Operator Interface is supplied with a Lithium Battery designed to maintain the internal memory and real-time clock during power outages. Assuming the operator interface terminal is powered up for 50% of the time, this battery should last over 4 years. A "Battery Low" system variable is available so that the programmer can choose specific action(s) to occur when the battery voltage drops below its nominal voltage.

It is possible to replace the battery without losing the contents of the Operator Interface's memory, but this does not reduce the importance of ensuring that a copy of the terminal's configuration is kept readily at hand to allow the terminal to be re-loaded in the case of mishaps. Please remember that it is not possible to extract the contents of a terminal for subsequent re-loading, so the importance of keeping a copy on disk cannot be over stressed.

## CHANGING THE BATTERY

To change the internal battery, follow these steps...

- ◆ Remove the power and PLC communications connector from the unit.
- ◆ Remove the four screws from the rear-cover.
- ◆ Remove the cover, ensuring that the right-hand edge is raised first to avoid fouling the earth stud. You may have to pivot the cover to an angle of about 30° to achieve this.
- ◆ If you wish to avoid losing the terminal's configuration, reconnect the power connector and re-apply power. Note that this will require the panel to be powered-up and, as such, only suitably qualified staff should carry out this procedure.
- ◆ The battery is located in a holder on the main circuit board. This should be clearly visible. Remove the battery from its holder.
- ◆ Place the new battery in the holder. The terminal's power supply can now be disconnected, if you re-applied power in the step above.
- ◆ Replace the lid, screws and connector by following the above procedure in reverse. You may like to make a note of the date the battery was replaced to allow planned maintenance to be carried out.
- ◆ If you did not keep the unit powered-up during battery replacement, hold down the EXIT and MUTE keys on the keyboard and remove and then re-apply power. This will clear the internal memory and thus the suitable configuration database should then be re-loaded.

Please note that the old battery must be disposed of in a manner which complies with your local waste regulations. Also, the battery must not be disposed of in fire or in a manner whereby it may be damaged and its contents come into contact with human skin.

## INSTALLATION & CONNECTIONS

The unit meets NEMA 4/IP65 requirements for indoor use, when properly installed. The units are intended to be mounted into an enclosed panel.

## INSTALLATION ENVIRONMENT

The unit should be installed in a location that does not exceed the maximum operating temperature and provides good air circulation. Placing the unit near devices that generate excessive heat should be avoided.

Continuous exposure to direct sunlight may accelerate the aging process of the bezel. The bezel should be cleaned only with a soft cloth and neutral soap product. Do NOT use solvents.

Do not use tools of any kind (screwdrivers, pens, pencils, etc.) to operate the keypad of the unit.

## MOUNTING INSTRUCTIONS

The Operator Interfaces are designed for through-panel mounting. A neoprene gasket is provided, to enable sealing to NEMA 4/IP65 specification. The panel cut-out diagram for the model supplied is provided. All mounting holes should be drilled for 0.14" clearance. Care should be taken to remove any loose material from the mounting hole to avoid such metal falling into the Operator Interface itself during installation.

## CONNECTING TO A PLC

The Operator Interface is designed to operate with a PLC. A serial communication connection must be made between the operator interface terminal and PLC, and the details of this connection vary according to which PLC is used.

The following section lists the connection details for the PLC to be used .

## PLC TYPE

Details on how to connect to most PLCs are available on request from RLC.

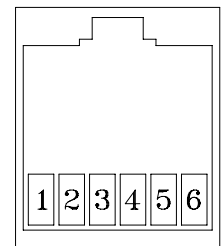
## CONNECTING TO AN IBM® PC/AT

The Operator Interface is programmed via software running on an IBM PC/AT or a compatible computer. The connection between the PC/AT and the operator interface terminal is made via a custom cable provided with the EDICT Developer's Kit. The cable is designed for a 9-way serial port. Please contact your supplier if you require a 25-way version.

## PROGRAMMING PORT PIN OUT

The Operator Interface's programming port is sometimes used to connect other RS-232 devices, such as printers. The following illustration and table gives the pin-out of this port to enable such connections to be made.

RJ11 FEMALE	
PIN	NAME
1	RTS
2	Tx
3	GND
4	GND
5	Rx
6	CTS



The above table denotes the pin names of the programming port. When connecting, the pin name at the programming port is connected to the opposite of that pin name at the destination device.



## What You Will Need

### Operating System

Your machine should be running Windows 3.1 or Windows 95, with the latter being the recommended operating system for this product. EDICT-97 will not operate with earlier versions of Windows. If you are using a recent version of Windows NT on an Intel machine, you should be able to run EDICT-97 without any problems, but this configuration is not supported at this date.

### Memory and Disk Space

You should have at least as much memory as is stated as the minimum requirement for your chosen operating system. Windows 3.1 users should operate in 386 Enhanced mode with a swap-file enabled, to ensure that virtual memory is available as back-up should it be required, Windows 95 users do not have to worry about this, as their virtual memory system is managed automatically.

EDICT-97 itself requires up to 4MB of disk space, and you should allow a further 250KB or so for every project you intend to create. Remember that Windows may need some temporary disk space for its own use, and you should always aim to keep 10MB or more space on your system disk. This is especially true if you intend to print to high resolution output devices like laser printers.

### Other Peripherals

EDICT-97 needs access to a serial port in order to download configuration information to an operator interface terminal. Remember that Windows does not allow you to unplug a mouse after the operating system has booted, and then use that serial port for some other purpose. Users with a single port and a serial mouse will thus have to boot Windows without the mouse installed, and manage without the mouse when using EDICT-97.

All of the facilities within EDICT-97 can be accessed without the use of a mouse or other pointing device, but we recommend that you use such a device to help you find your way around the software. That said, we also suggest that you try and learn the short-cut keys for common operations, as you will soon find yourself using the software more fluently as a result.

### Installing EDICT-97 from a Floppy Disk

EDICT-97 is supplied on three 3½" high density floppy disks. If you are unable to read such disks, you should consult your distributor about the availability of alternative formats. EDICT-97 is not copy protected, and you can install any number of copies from a single set of installation disks. However, your license agreement does not permit the software to be used outside a single organization, and you should ensure that you do not breach the terms of this agreement.

### Installing from Windows 3.1

To install from within Windows 3.1, follow these steps...

- Place Disk 1 in the appropriate drive, hereinafter assumed to be drive A
- Select the Run option from the File menu of the Program Manager
- Type a: \setup into the dialog box, and press the ENTER key.

Follow the instructions provided by the installation program.

## Installing from Windows 95

To install from within Windows 95, follow these steps...

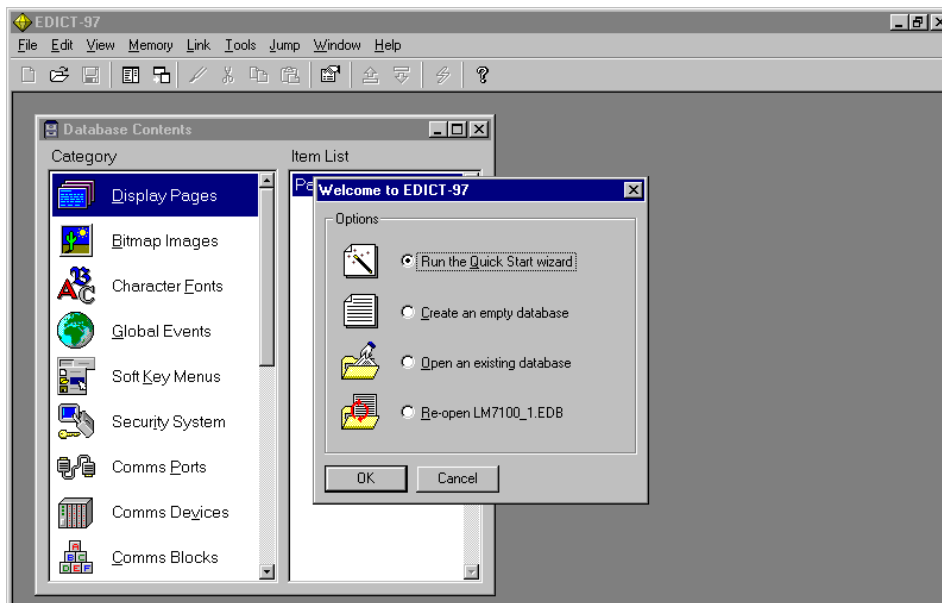
- Place Disk 1 in the appropriate drive, hereinafter assumed to be drive A.
- Select the Run command from the menu on the Start button.
- Type a: \setup into the dialog box, and press the ENTER key.
- Follow the instructions provided by the installation program.

## Connecting your PC to your HMI

Once you have successfully installed EDICT-97 software to your computer, you will need to connect your PC to the Paradigm HMI product you have purchased. The PC to Paradigm connection is easily made via the programming cable (p/n P890301Z). This programming cable was included in your EDICT-97 development kit. Additional cables may be purchased from your Red Lion Controls distributor.

## Using EDICT-97

Double click the EDICT-97 icon on your computer. The following screen should appear on your PC screen.



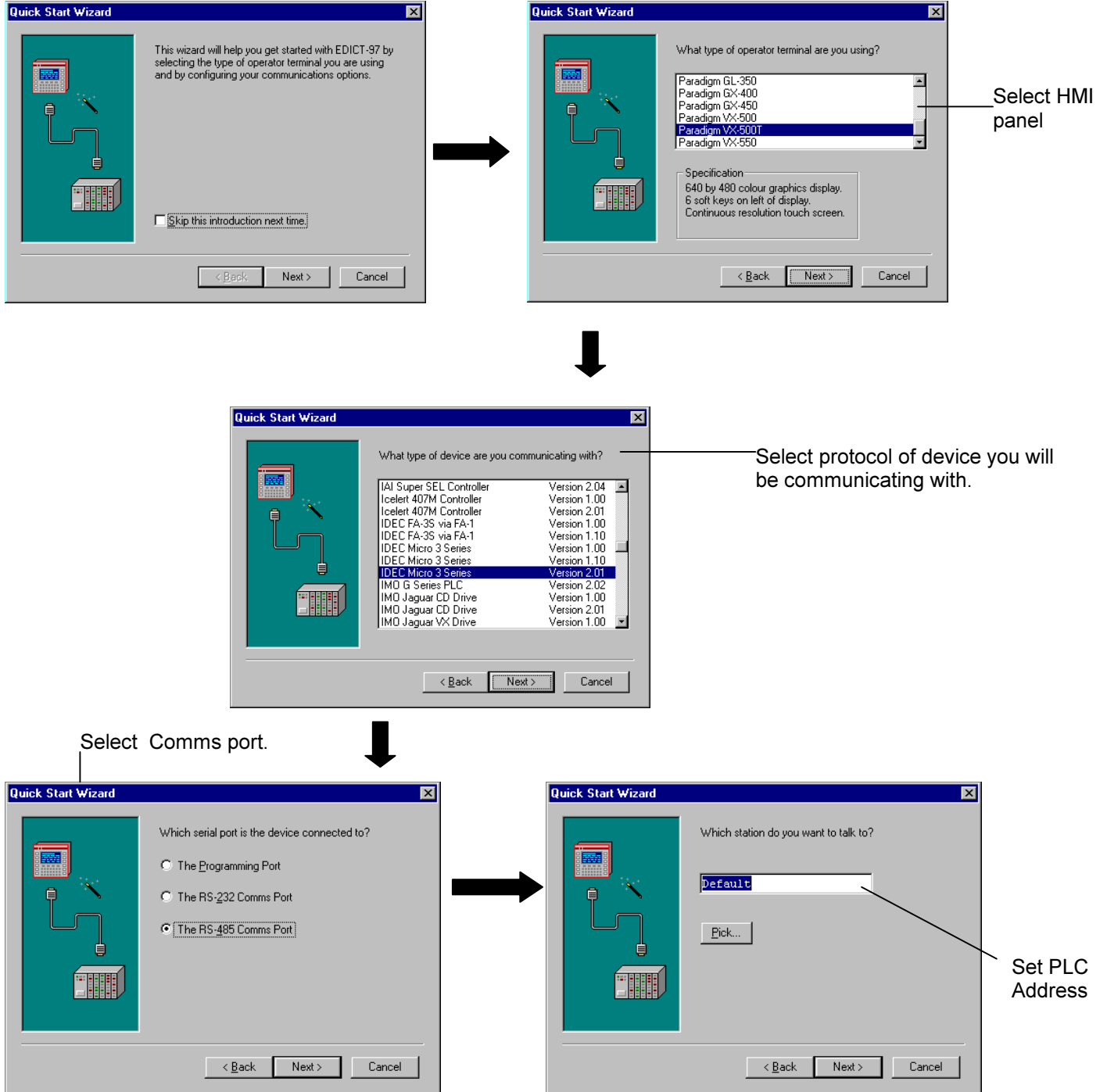
At start up, EDICT-97 will give the user 4 options. They are Run the Quick Start wizard, Create an empty database, Open an existing database or Re-open the last database configured in EDICT-97.

## A Quick Start Guide for writing an EDICT-97 database

Use the Quick Start Wizard that appears when you start EDICT-97.

(Make sure you are using the correct Paradigm/PLC communication cable for the PLC selected)

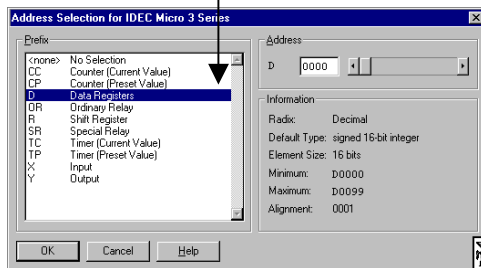
This will allow you to select the HMI panel you will be using as well as setting up your communication options.



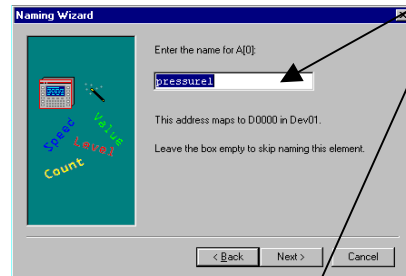
To complete the communication link **Jump to CommsBlocks** and select the registers you are accessing in the device you have selected..

Communications Blocks						
Device	Address	Data Type	Size	Access	Update	
A	WizPort3	D0000	16-bit Signed	2	Read	Auto
B	None	None	16-bit Signed	0	Read	Auto
C	None	None	16-bit Signed	0	Read	Auto

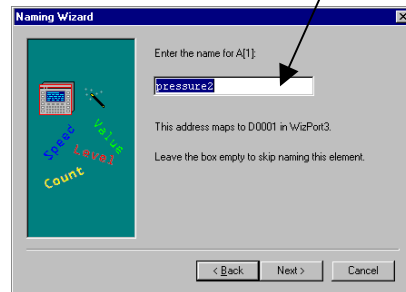
Select registers you want to access in the PLC.



The Named Data wizard allows you to rename the registers.

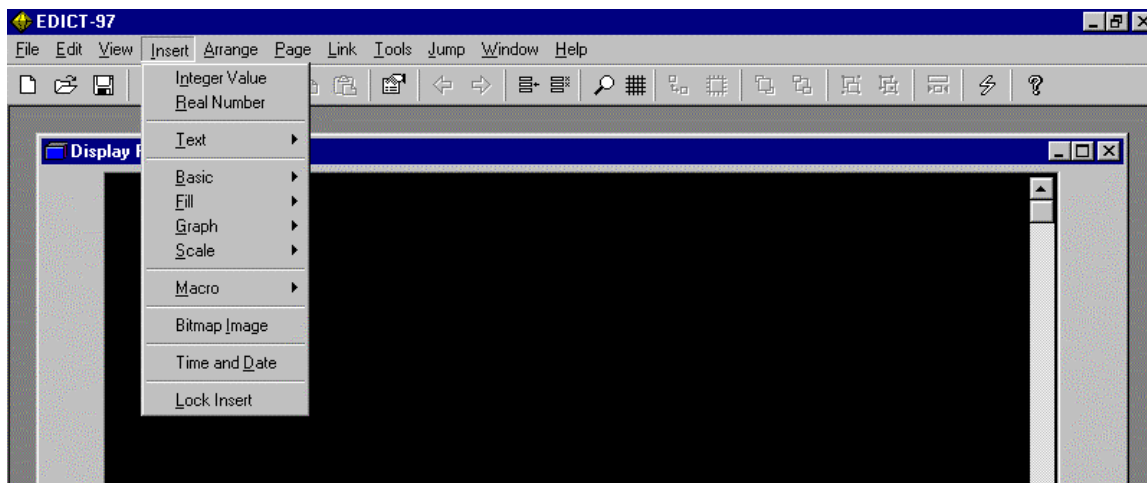


Now the PLC registers D0000 and D0001 can be referred to as pressure1 and pressure2 respectively.

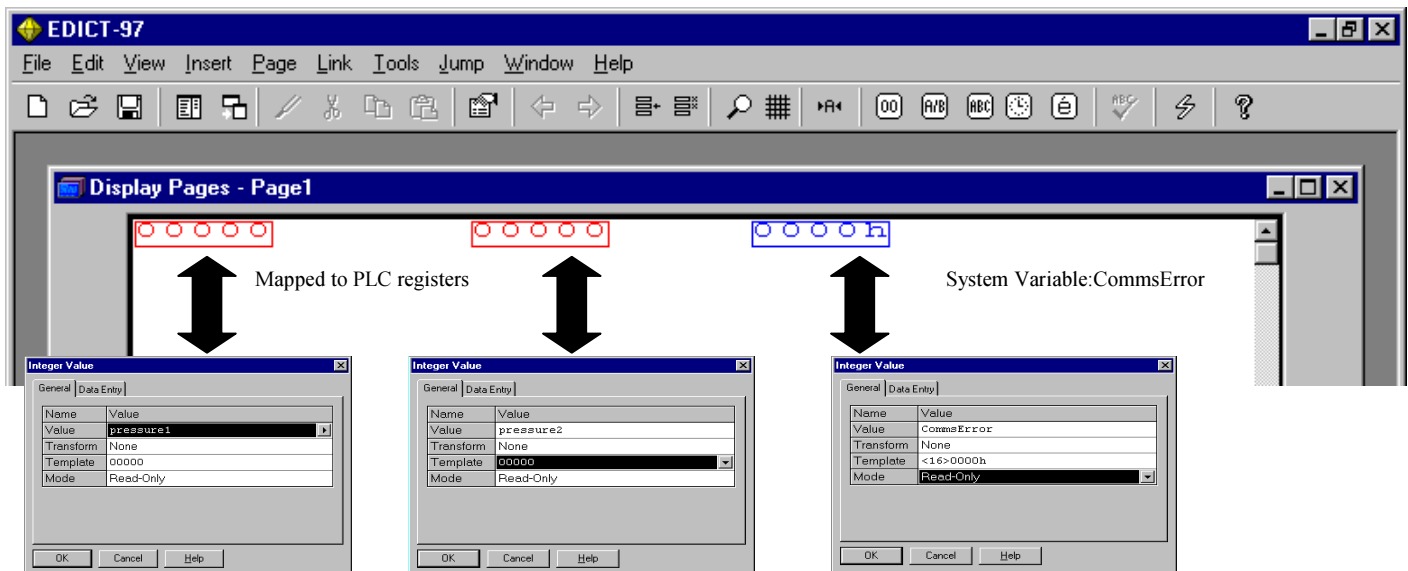
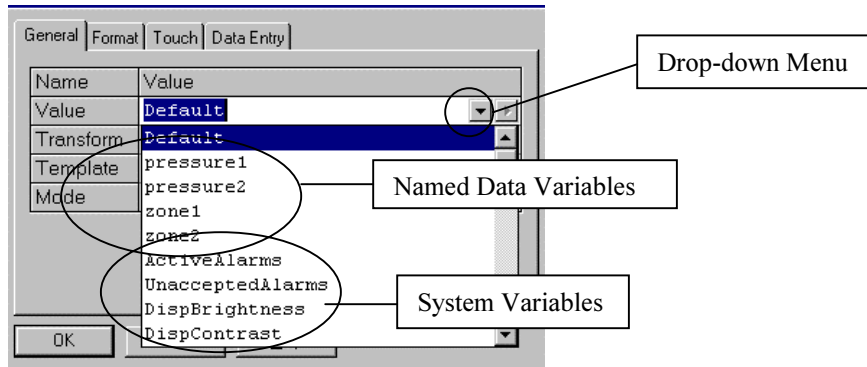


To confirm that your communications are properly configured **Jump to Display Pages** and open Page1. Next insert the following on Page 1.

Select **Insert Integer Value** from the EDICT-97 Toolbar.



You can use the drop down menu to select Named Data Variables or System Variables.



**CommsError:** A system variable that can be inserted as an Integer Value of a display page. Use the <16>0000h hexadecimal template for this variable. A display value on the display page of 0000h for this field indicates no problem with communication. The least significant bit for this variable going high indicates a Communication Error. The next significant bit going high indicates that the Error is with Device 1(0003h). A communication Error with Device 2 will result in a displayed value of 0005h for this field,etc. **See Page D7 of the EDICT-97 manual for details on Troubleshooting Communications.**

It is suggested that you do not proceed beyond this point until you have established communications between the PLC and your Paradigm HMI panel.

### EDICT-97 is an event driven software program.

Designers can choose from a long list of events (See page 54 of the EDICT-97 Software manual) to provide the desired action (See Section A of the EDICT-97 Software manual for details).

**Events:** There are 2 types of events in EDICT-97. They are local events and global events.

**Local events** (and their programmed Actions) occur only when the display page they are programmed on is currently displayed on the programmers HMI.

	Event	Enable	Action	Routing
1	Soft-key 1 pressed	Default	GotoPage (Page2)	Default
2	None	Default	None	Default
3	None	Default	None	Default
4	None	Default	None	Default
5	None	Default	None	Default
6	None	Default	None	Default
7	None	Default	None	Default
8	None	Default	None	Default

In the example above **Soft-key 1 pressed** (type directly or use the Event pull down window) will cause the HMI panel to **Go to Page 2**. The Action can be typed directly into the Action window or the Action Wizard will help the programmer select the desired action when the window is expanded. This Event will cause this Action to occur on this display page only.

**Global events** (and their programmed Actions) occur regardless of the page currently displayed on the programmers HMI. To create a Global event **Jump to Global Events**.

	Event	Enable	Action	Routing
1	PREV key pressed	Default	GotoPrevious ()	Default
2	None	Default	None	Default
3	None	Default	None	Default
4	None	Default	None	Default
5	None	Default	None	Default
6	None	Default	None	Default
7	None	Default	None	Default
8	None	Default	None	Default
9	None	Default	None	Default
10	None	Default	None	Default
11	None	Default	None	Default

In the example above **PREV key pressed** (type directly or use the Event pull down window) will cause the HMI panel to The Previously displayed page regardless of the page currently displayed on the HMI.

Note: For an explanation on **Enable** and **Routing** usage see the EDICT-97 Software manual.

2) This section deals with using Alarms in your database.

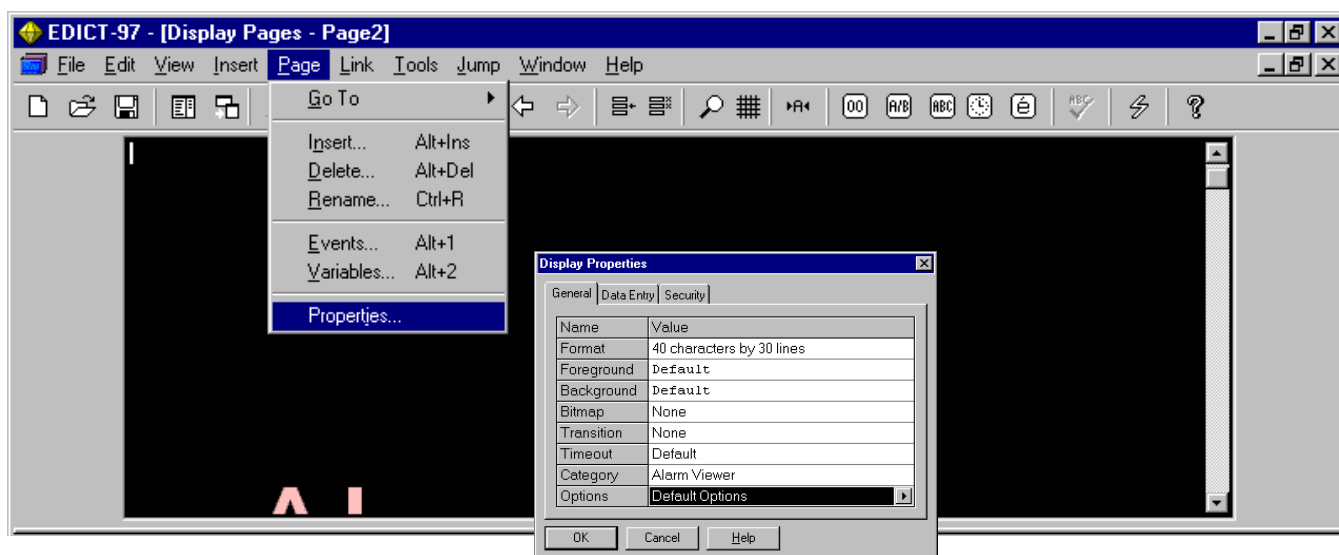
### Jump to Alarm Scanner

Note: for this example 2 variables were used zone1 and zone2. These variables are reading temperature values from a Red Lion Controls PID temperature controller.

	Alarm Name	Expression	Trigger	Accept	Priority
1	zone1 high temp	zone1>300	High	Auto	0
2	zone2 high temp	zone2>300	High	Manual	0
3	Shut down warning	zone1>400   zone2>400	High	Manual	0

Alarm names and the Expressions which define the alarms are configured in the Alarm Scanner category of EDICT-97. For this example both zone1 and zone2 trigger above 300. The Shut down warning alarm will trigger when either zone1 or zone2 is above 400. ("|") is the logical or operator. See operators in Section A of the EDICT-97 software manual.)

To view the alarms create a display page and edit the page properties so that this page is an alarm viewer.



Use Global events to program the Mute Key (Mute Siren()) and Alarms Key (GotoPage(Alarms)) to mute and view alarms.

3) This section deals with using the Trigger Table in your database.

#### Jump to the Trigger Table

EDICT-97 - [Trigger Table]			
Expression	Edge	Action	
1 pressure1>300	Rising	ModemDial(1, "8005551234")	
2 Default	Rising	None	
3 Default	Rising	None	
4 Default	Rising	None	

The expressions in the Trigger table are evaluated after every Comms Update. If the expression is "True" then the programmed action is triggered. For the example above the number will be dialed when the variable pressure1 is greater than 300.

4) This section deals with using the Schedule Table in your database.

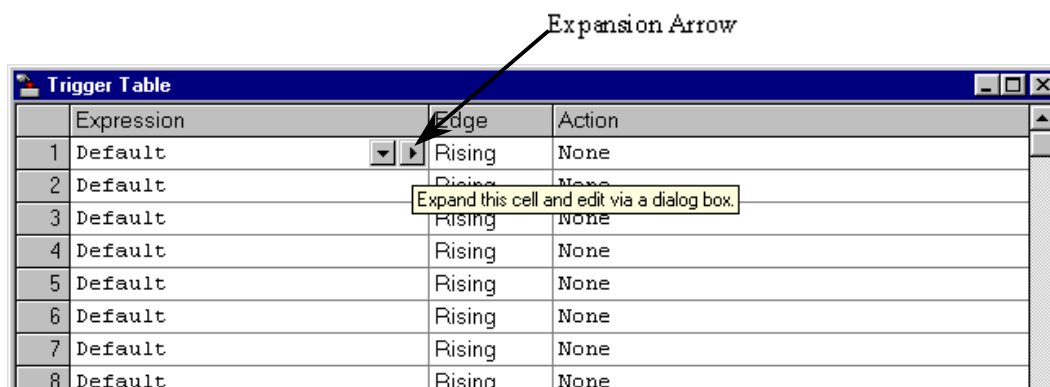
#### Jump to the Schedule Table

EDICT-97 - [Schedule Table]					
	Days	Hour	Min	Sec	Action
1	Mon	07	00	00	zone1:=100
2	Weekdays	??	00	00	PrintReport(1, HourlyRate)
3	None	00	00	00	None
4	None	00	00	00	None
5	None	00	00	00	None

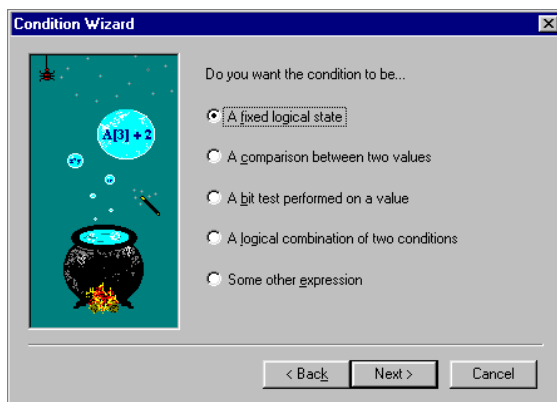
The Schedule table uses EDICT-97's real time clock to schedule programmed actions. For the previous example the variable zone1 will be loaded with the value of 100 on Monday 7:00AM. In addition the report "HourlyRate" will be printed every weekday at the top of each hour. See the EDICT-97 software programming manual for more details.

## Using the Condition Wizard...

In this example, a variable, called Temperature, will be evaluated to determine if it is less than or equal to 100. First, the expansion arrow is selected in the expression column of the Trigger Table.



The Condition Wizard appears (if the wizard introduction is turn on, this screen will be first, click Next to get to the following screen)...



**A fixed logical state** allows the insertion of TRUE or FALSE in the expression column. TRUE is equal to a value of 1 and FALSE a value of 0. TRUE would cause a continuous action, whereas FALSE would cause the action to never occur.

**A comparison between two values** executes an evaluation. As per our example, this selection would allow us to insert Temperature <= 100. Then if the temperature falls to 100 degrees or below, the action will occur.

**A bit test performed on a value** will evaluate a specified bit of a value. The user has the choice to evaluate the bit as on or off. Whenever the condition is true, the action will be performed.

**A logical combination of two conditions** is used for inserting multiple condition to evaluate before an action will occur. To expand our example, if we wanted to have the action occur when the temperature falls to 100 or below or when the pressure falls below 50 psi this selection would allow us to create and insert the syntax needed. The syntax would look like the following when finished: Temperature <= 100 || Pressure < 50.

**Some other expression** allows the insertion of many types of variables. These include Direct PLC references, Comms Blocks items, data variables, bits, constants, math functions (add, subtract, multiply, divide, etc), and Edict function calls.

As mentioned, this example calls for selecting **A comparison between two values**. Click Next.

The following screen allows for the creation of our desired expression. In this case, Temperature <= 100.

The 'Condition Wizard' dialog box is shown. It has a title bar with a close button. On the left is a decorative image of a cauldron with a flame and a bubble containing 'A[1] + 2'. The main area is titled 'Define the comparison for the condition...'. It contains three input fields: '1st Value:', 'Comparison:', and '2nd Value:'. Each field has a 'Pick...' button to its right. The 'Comparison:' field is a dropdown menu currently showing 'Is Equal To'. At the bottom are three buttons: '< Back', 'Next >', and 'Cancel'.

The Pick buttons aid in selecting Direct PLC references, Comms Blocks items, data variables, bits, constants, math functions (add, subtract, multiply, divide, etc), and Edict function calls for each operand in the expression. Our first value, or operand, will be a data variable called Temperature. The second is a constant, 100. Our operator selection is for Is Less Than or Equal To. The following should be how this screen should look before clicking Next.

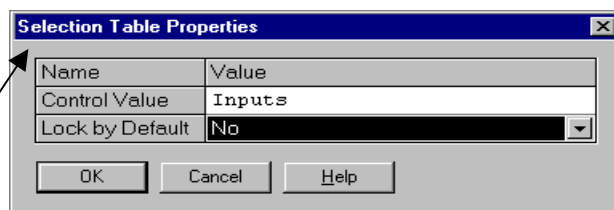
The 'Condition Wizard' dialog box is shown again, but now the fields are populated. The '1st Value:' field contains 'Temperature', the 'Comparison:' dropdown shows 'Is Less Than or Equal To', and the '2nd Value:' field contains '100'. The 'Pick...' buttons are still present. The 'Next >' button is now highlighted with a dotted border, indicating it is the recommended next action.

Clicking Next shows the Wizard Summary, reiterating the components and the expression to be created. Clicking Done will insert the expression in the Trigger Table.

Trigger Table			
	Expression	Edge	Action
1	Temperature <= 100	Rising	None

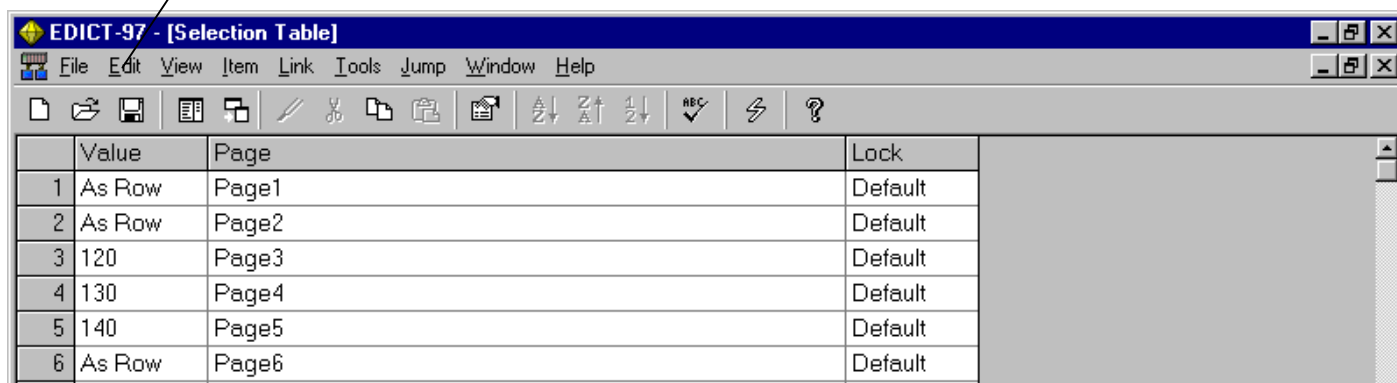
5) This section deals with using the Selection Table in your database.

#### Jump to the Selection Table



The dialog box titled "Selection Table Properties" contains three fields: "Name" (empty), "Value" (containing "Inputs"), and "Lock by Default" (a dropdown menu set to "No"). At the bottom are "OK", "Cancel", and "Help" buttons.

The control value is the register that controls which display page is on the screen of the HMI. "Inputs" is the named data variable for the input register of this PLC.



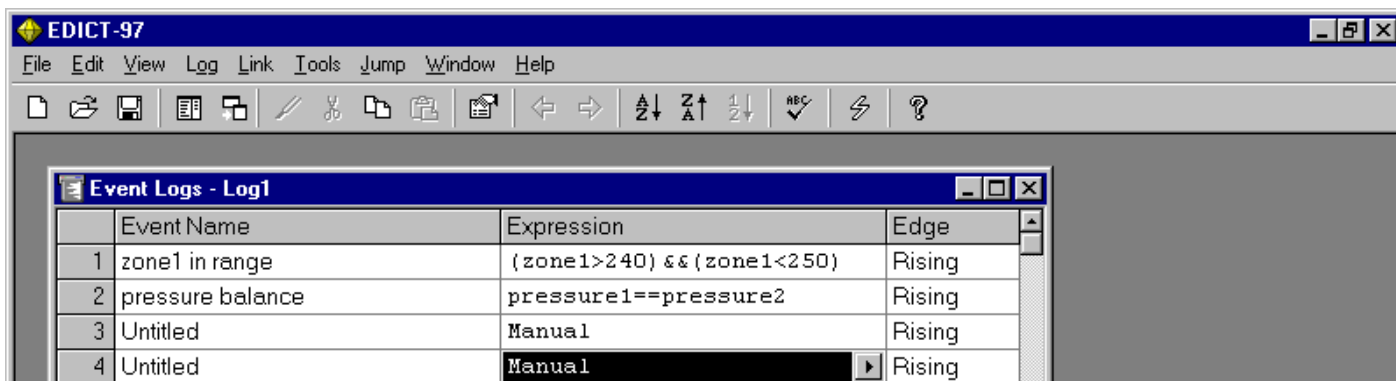
The window shows a table with 4 columns: Value, Page, and Lock. The table contains 6 rows of data.

	Value	Page	Lock
1	As Row	Page1	Default
2	As Row	Page2	Default
3	120	Page3	Default
4	130	Page4	Default
5	140	Page5	Default
6	As Row	Page6	Default

For this example Page1, Page2 and Page6 will be displayed if the value of "Inputs" is 1, 2 or 6 respectively. Page3, Page4 or Page5 will be displayed if the value of "Inputs" is 120, 130 or 140 respectively. Any register can be used as the Control value register. See the EDICT-97 software programming manual for more details.

6) This section deals with using the Event Logs in your database.

#### Jump to the Events Logs



The window shows a table with 4 columns: Event Name, Expression, and Edge. The table contains 4 rows of data.

	Event Name	Expression	Edge
1	zone1 in range	{zone1>240} && {zone1<250}	Rising
2	pressure balance	pressure1==pressure2	Rising
3	Untitled	Manual	Rising
4	Untitled	Manual	Rising

The EDICT-97 Event Log allows the programmer to keep a record of events happening (time of event, start time of event and end time of event). In the preceding example the event “zone1 in range” is activated when the zone1 temperature is between 240 and 250 F. The second event “pressure balance” occurs when the value of pressure1 is equal to pressure2. To view the events log **Jump** to a display page and set the page properties to **Event Viewer**.

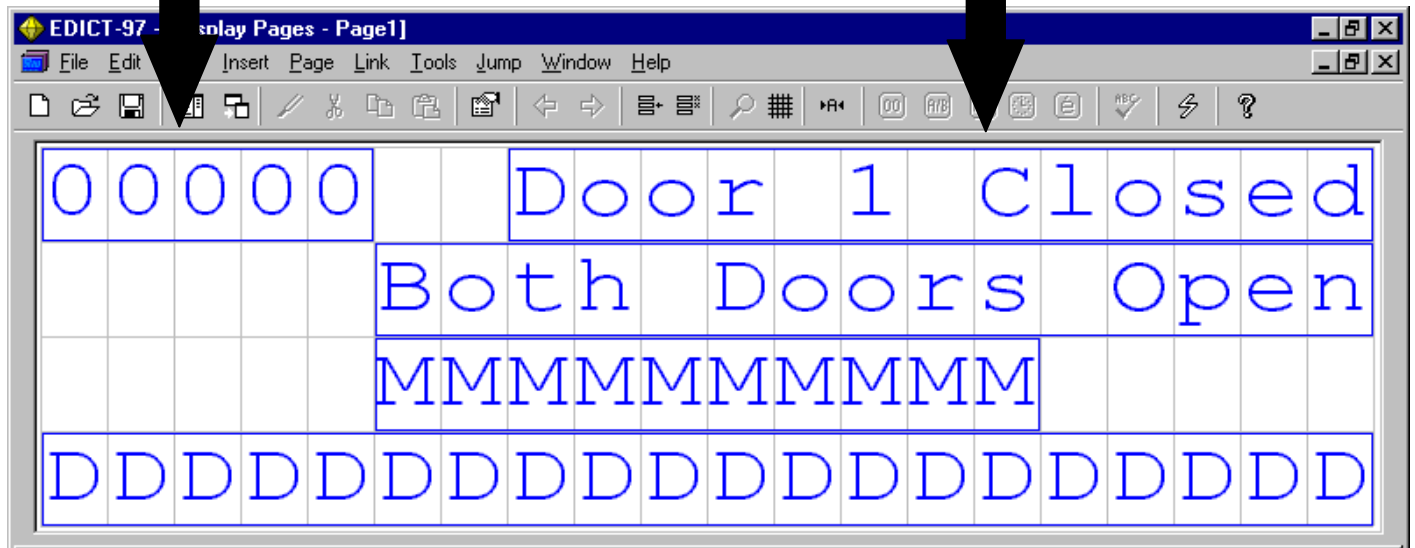
## Jump to a Display Page

**Status Text** [X]

General | Data Entry

Name	Value
Value	Inputs.0
ON Text	Door 1 Open
OFF Text	Door 1 Closed
Length	Automatic
Mode	Read-Only

OK Cancel Help



See below for the Quad Text, Message Text and Decode Text windows for this application.

**Quad Text**

General | Data Entry

Name	Value
Value	Inputs
Text 0	Doors Closed
Text 1	Door 1 Open
Text 2	Door 2 Open
Text 3	Both Door 1 and Door 2 Open
Length	Automatic
Mode	Read-Only

OK Cancel Help

← Quad Text Example

**Local Message Table**

	Text
1	Door 1 Open
2	Door 2 Open
3	Door 1 and Door 2 Open
4	Door 3 Open
5	Door 3 and Door 1 Open
6	Empty Message
7	Empty Message
8	Empty Message
9	Empty Message
10	Empty Message
11	Empty Message
12	Empty Message

OK Cancel Help

← Message text from Local Message table. This message table is valid only on this display page. EDICT-97 also has a Global Message Table. This message table can be accessed from any display page in your database. To create a global message table **Jump to Message Table**.

**Decode Text Table**

	Condition	Text
1	Inputs.0==1&run==1	Machine running/door1 open
2	Inputs.0==0&run==1	Maching running (All doors closed)
3	Inputs.0==1&run==0	Machine Stopped (All doors closed)
4	Default	Empty Message
5	Default	Empty Message
6	Default	Empty Message
7	Default	Empty Message
8	Default	Empty Message
9	Default	Empty Message
10	Default	Empty Message
11	Default	Empty Message
12	Default	Empty Message

OK Cancel Help

← Decode Text example. The Decode text table will evaluate the conditions from top to bottom ( 1 to n). The text for the first true condition will be displayed.

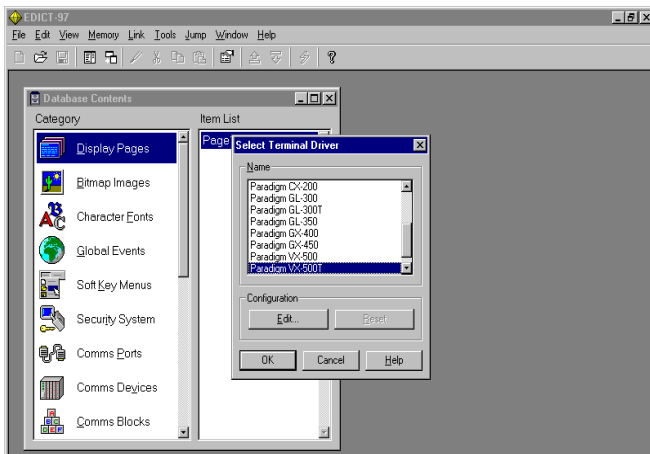
Note: The Wizard will configure the communication settings for one device. If you are communicating with more than one device, you will need to configure **CommsPorts** and **CommsDevices** for the additional devices.

### Configuring Without the Quick Start Wizard

The first step to setting up your database requires that you specify the Paradigm unit you are communicating with.

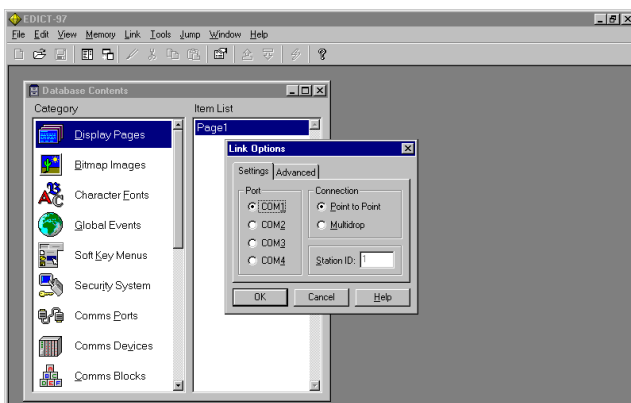
From the top line on your EDICT-97 software choose **File**, and then choose **Terminal Selection**. The following screen will appear on your PC.

Scroll down and choose the Paradigm unit you are communicating with and hit **Enter**.



The next step involves choosing the serial communication port on your PC. From the top line of your EDICT-97 software choose **Link**, then choose **Options** and the following screen should appear.

Chose the appropriate serial port on your PC and hit **Enter**.



# Downloading

The process of sending a configuration database to an operator terminal is known as downloading. EDICT-97 uses a technique known as “incremental download” to avoid sending data which has not changed since the download was last performed. This technique results in greatly reduced download times, and makes it much easier for you to perform small modifications to a database and then to test the results.

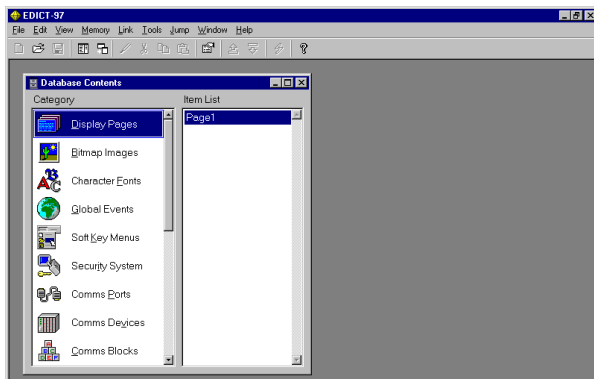
To download the current database to a terminal, connect a programmable cable from a spare COM port on your machine to the programming port of the terminal. You should check that EDICT-97 is configured to use the correct COM port by means of the “Options” command on the “Link” menu. You can then send the database by selecting the “Update” command from the same menu, or by clicking on the “lightning-bolt” button on the toolbar. The same result can be achieved from the keyboard by pressing the F9 key.

EDICT-97 will sometimes send the entire database if it finds that the database in the terminal makes incremental download impossible. You can force EDICT-97 to do this by using the “Send” command instead of the “Update” command, but this should not be necessary unless you want completely to clear the contents of terminal. All Comms blocks and other internal storage areas are set to zero when a complete download is performed.

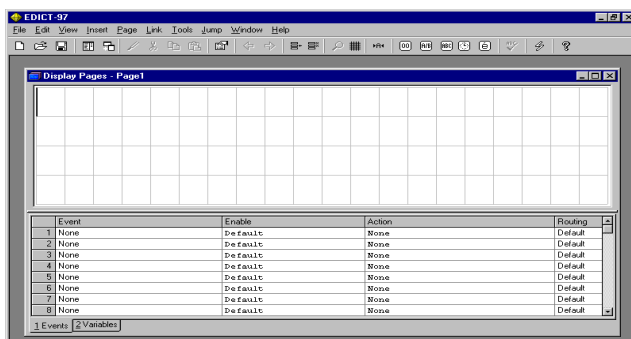
If you find you are unable to download to a terminal, it is possible that the programming port of the device has been configured for use by a Comms driver and is thus no longer available for programming purposes. The designer of the terminal’s database should have provided a way to stop the system and allowing programming, typically by calling the “StopSystem” function in response to some key sequence. If this facility is not available, you will have to clear the terminal’s memory before you can download. Since clearing the memory negates the point of the incremental download facility, you should always try to provide a way of calling “StopSystem” if circumstances make it necessary.

## You are now ready to create a Test Database

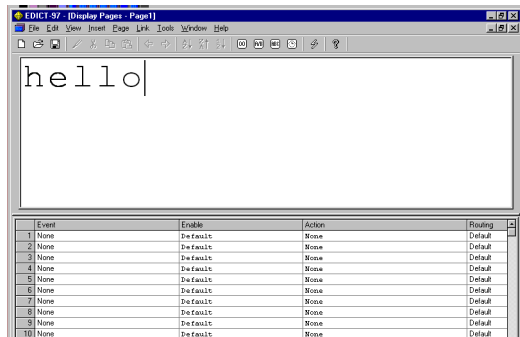
To test the programming link between your PC and the Paradigm unit, you need to create a simple test database that you can download. Choose **Display Pages** from the Category column of the EDICT-97 Software. After you choose **Display Pages**, the following screen will appear.



Notice that an Item list appears on the right hand side of your software screen. Choose **Page 1** and the following screen will appear.



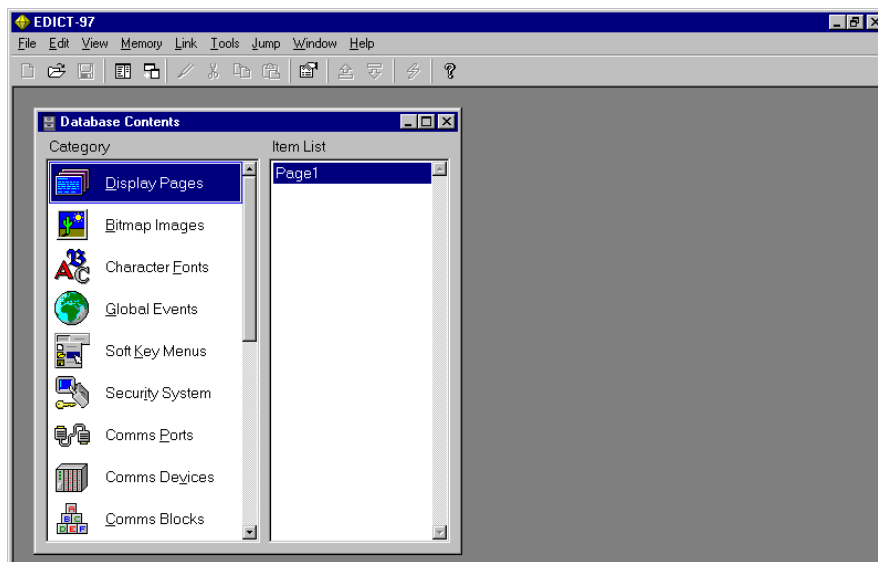
The blinking cursor shows your position on the page. Type in some sample text like “hello”.



To download the test database, hit the **F9 function key** at the top of your keyboard or click the lightning bolt icon on your toolbar (next to the question mark). A successful download will result in **hello** being displayed on your HMI display. If you were not successful, check for proper Com Port designation (**Link** then **Options**).

## A QUICK START OF EDICT-97

### The Database Contents Window



As you will have noticed while preparing your test database, EDICT-97 starts up by displaying the Database Contents window. This window lists the various items that make up the database, and is used to select the item with which you wish to work. It can also be used to manipulate the lists use to store named items, such as display pages and user programs. Pressing the **CTRL+1** key combination at any time will close all other windows, and return you to the database contents window.

The left-hand half of the window contains an icon for each area of the database. If the selected icon refers to a list of named items, the right-hand half of the window will show the list of items, with the current selection highlighted. Double-clicking in either half of the window will open the item in question, as will pressing the **Enter** key. You can move between the sections of the window by using the **Left** and **Right** keys. Navigation within either of the lists is performed in the usual way, using the arrow keys or the mouse. When working with the icon list, you may also press the underlined character of a given entry to select that entry. Each entry has a unique letter associated with it, and these are well worth learning.

## **The 19 Categories (sections) in the Database Contents Window:**

### **Display Pages**

This is where each display page is created. A display page is what appears on your HMI at any particular time. A display page may contain text, messages, data (both read and read/write), time/date and results of calculations performed in EDICT-97. Typical user applications can contain up to several hundred display pages.

### **Bitmap Images (Graphic units)**

For importing Bitmaps into EDICT-97 which will be displayed on our monochrome and color Graphic units. Bitmaps are displayed on the Graphics or Bitmap layer of a display page.

### **Character Fonts (Graphic units)**

For importing Fonts into EDICT-97 which will be displayed on our monochrome and color Graphic units. Imported Fonts are displayed on the Graphics layer of a display page.

### **Global Events**

There are events that occur (e.g. Function 1 Key being pushed on your HMI) that can cause a desired action (e.g. Go to Previous Page). Global Events are events that give the same desired action regardless of page. Local events (set up in Display Pages) give the desired action only on that specific page.

### **SoftKey Menus**

EDICT-97 has the ability to create Soft Key menus. These menus will slide out on a display page when an event triggers the action ShowMenu(). The number of Soft Key Menus that can be created is limited only by the amount of memory available in the designer's database.

### **Security System**

EDICT-97 features a fully configurable, hierarchical user specific access control scheme. Access can be controlled at both the page level and at the animation level.

### **Comms Ports**

There are 2 or 3 communication ports (2-RS232; 1-RS-485) in every Red Lion Controls' HMI. This section is used to set up their designated function. For example, the RS485 port may be designated for **Allen Bradley SLC PLC**.

### **Comms Devices**

In this section you can program up to 29 Comms Devices. This allows a great deal of flexibility when setting up your communication schemes. For example Port 3 (RS485) may be set for the Red Lion Controls driver. If you had 10 Red Lion Controls instruments with RS485 communication ports, you could easily set up Device1 through Device10 to access each of these instruments.

### **Comms Blocks**

EDICT-97 allows you to define up to 26 communications blocks, each of which is given a name comprising a single letter. Each block may be used to transfer a given region of PLC memory to or from the HMI. A block has a number of properties, which are defined using the table within the Comms Blocks window.

e.g. Comm Block A[ ] may be set up for 2 counter values from a PLC . A[0] = value of counter 1 (value =200), A[1]= value of counter 2 (value =400)

### **Named Data**

EDICT-97 allows you to apply names to either existing expressions or to internal memory locations. Using the Named Data section declares the named data item globally. It can also be set up as a local named data item in Display Page section. Named data items are divided into four categories: Variable, Constant, Formula and Data File.

e.g. of variable "AVG"

AVG:  $= (A[1] + A[2]) / 2$  Using the counter values above AVG would return a value of 300 (i.e.  $(200 + 400) / 2 = 300$ )

### **Alarm Scanner**

EDICT-97 contains an alarm scanner, which continually monitors up to 500 alarm points defined in the alarm table. Each alarm point typically has a triggering expression, the state of which determines the conditions under which the alarm will be triggered. You can also indicate the mechanism to be used to accept each alarm, and whether a given alarm point will activate the HMI's internal sounder or not. Priority assignment allows for greater flexibility.

### **Trigger Table**

EDICT-97's trigger scanner continually scans up to 500 entries in the trigger table. Each entry has a controlling expression and an edge type. When the expression changes in the direction specified by the edge type, EDICT-97 will execute the associated action. The action may cause a new page to be displayed, or may perhaps change a value in a remote device.

### **Schedule Table**

EDICT-97's real-time schedule continually scans up to 500 entries in the schedule table, matching the conditions described in each entry against the current time and day of the week. When all of the specified conditions are met, the action associated with that entry is executed. This action may select a new display page, change a value in a remote device, or perhaps print a shift report.

### **Selection Table**

This window is used to edit the properties of the selection table, which allows a remote PLC or similar device to force a particular page to be displayed. The table will not be displayed until you have edited the table properties to indicate which data value should be used to select a given page. To do this, select the "Properties" command from the "Item" menu, and edit the "Control Value" property. The Selection Table contains 500 points, each of which has a number of properties, represented by the columns of the table. Please follow the link below for information about the properties of each selection table entry.

### **Message Table**

EDICT-97 contains a global message table capable of storing up to 500 messages.

**Data Logger (Graphic units)**

The Data Logger allows designers to display a graphical trend of variables versus time. Up to 500 variables can be logged.

**Event Logs**

EDICT-97 contains an Event Log, which continually monitors up to 500 Events defined in the Event Logs table. Each Event point typically has a triggering expression, the state of which determines the conditions under which the Event will be triggered.

**Printed Reports**

The printed report editor is used to define full-page reports, which can then be sent to a printer attached to one of the terminal's Comms ports. Each report may contain a combination of static text and animation items.

**Programs**

To provide the ultimate in flexibility, EDICT-97 contains a programming language, similar to the "C" and "Java" languages used in so many applications. Each program is equivalent to single function within these languages, and may perform anything from a simple list of actions to a complex combination of decision-making or loop constructions. Do not be too alarmed if the prospect of writing programs in either of these languages seems rather too much to handle, as most applications can be handled with either no programming, or with very simple programs which simply comprise action lists.

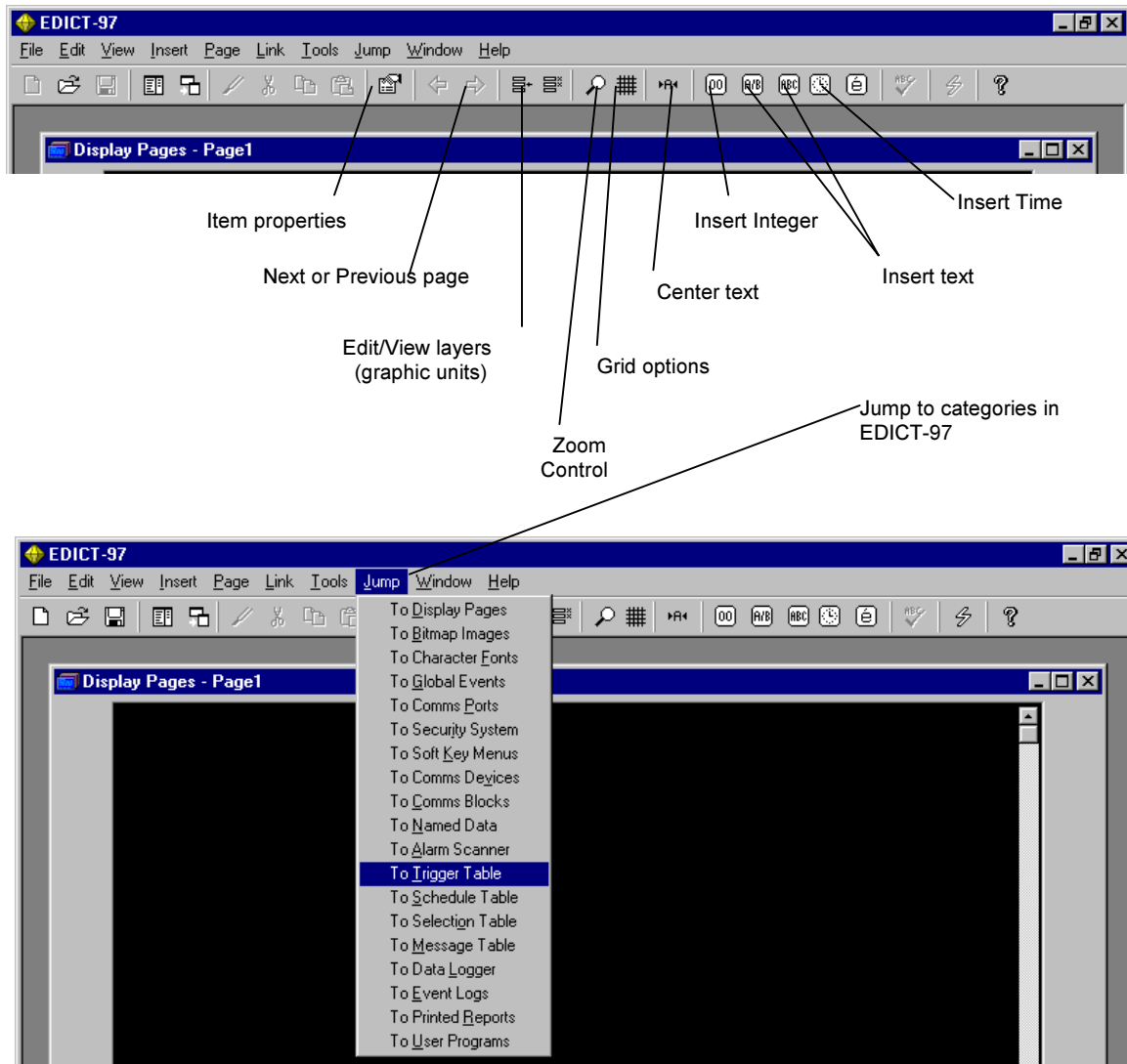
# Working with EDICT-97

## Section 1 : Display Pages

Display pages are used to define what should be shown on the operator terminal's display at any given time. Each page may contain a combination of static text and animation items. There are various types of animation items, each capable of representing plant or internal data in a specified way. Certain animation items can be configured for data entry, allowing the operator to modify plant settings or internal data items.

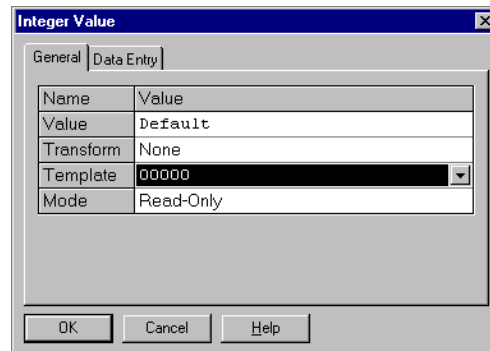
Each display page also has an associated event map, which specifies what should happen if a given event occurs when the page is displayed. As an example, an event is generated when a key is pressed, and an entry in the event map can be used to change data or to select another page in response to that event. Events also occur in response to changes within EDICT-97's internal state, and can be used to customize the terminal's behavior.

### The Tool Bar Icons



## Insert Integer Value

Choose this Icon and the following appears

The image shows a dialog box titled "Integer Value" with a close button (X) in the top right corner. It has two tabs: "General" and "Data Entry". The "Data Entry" tab is selected. Inside the tab, there is a table with four rows: "Name" with value "Value", "Value" with text input "Default", "Transform" with a dropdown menu showing "None", and "Template" with a dropdown menu showing "00000". The "Mode" row shows "Read-Only". At the bottom of the dialog are three buttons: "OK", "Cancel", and "Help".

Property	Description
Value	An integer expression giving the value to be displayed. If the field is set to "read-only" operation, any expression can be used, including those involving scaling or other processing. If "Data Entry" mode is selected, a writable value must be entered.
Transform	An optional transform to be performed on the data. Before the data value is displayed, it will be transformed using the information in this property. This can be used, for example, to scale values from the PLC to engineering units. Note that the "Value" property is always evaluated as a 32-bit number, and transforms that manipulate bits will thus behave as if they have been passed a 32-bit argument, even if the underlying data is really a 16-bit value.
Template	A picture of how the number should be formatted. You should use this field to select the number of decimal places required and the number base to be used. You may either select a value from the list, or enter a custom setting of your own.
Mode	Whether or not data entry is to be enabled. If this property is set to "read-only", the animation item will simply display the expression given by the "Value" property. Selecting "Data Entry" will allow the operator to enter a new value by using the numeric keys or the Raise and Lower buttons.

## Data Types

The compiler used within EDICT-97 supports a number of data types, including a wide variety of integer types, a floating point type and a dynamic string type. The floating point type uses 32-bit IEEE representation to hold values with an accuracy of around 7 significant figures. Stored string variables can be up to 256 characters in length, although intermediate values may exceed this length considerably. The table below lists the integer data types, together with the range of value each can hold.

Type	Range
16-bit Unsigned Value	0 to 65535
32-bit Unsigned Value	0 to 4294967295
16-bit Signed Value	-32768 to +32767
32-bit Signed Value	-2147483648 to +2147483647

In general, EDICT-97 will look after conversions between types as and when required by the context, and will automatically “promote” a data value to the next larger type should the current type prove too small to hold an intermediate value during a calculation. If you need to perform an explicit type conversion, you can use what is known as a type cast sequence.

The best way to get an idea how data is entered is to do some simple examples.

### **Example: Adding an Integer Animation Item**

**Step 1)** If for example you wanted to insert a 2 digit Read/Write Integer value on your display page. You would enter the following.

Name	Value
Value	A[0]
Transform	None
Template	00
Mode	Data Entry

Note: This value is designated to go into CommBlock A[0].

**Step 2.** In addition, I want to restrict the value of the number entered to be in from 0 to 20. To accomplish this, choose **DataEntry**, and enter the following (Min =0, Max =20).

Name	Value
Minimum	0
Maximum	20
Step	1
Default	Default
Enable	Default
Access Level	Any Level
Touch Entry	Default

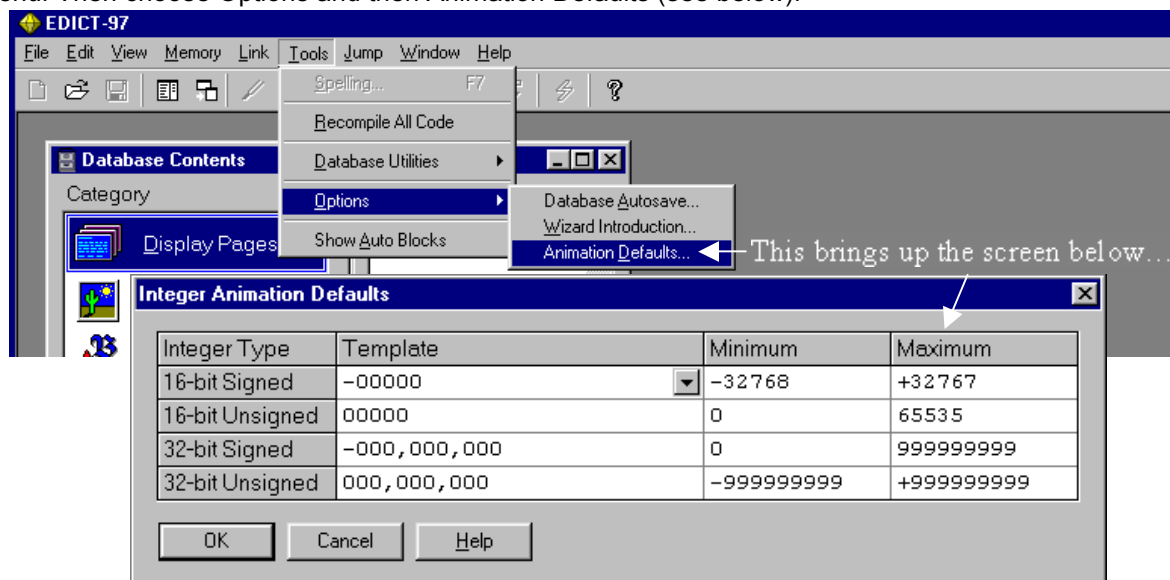
### **Data Entry Properties**

When a field is enabled for data entry, the properties contained within the second tab of the associated dialog box may be used to control how data entry will occur. The table below lists these properties, not all of which will be available for all animation types.

Property	Description
Minimum	The minimum value permitted. If an expression is entered, the value entered by the operator must not be less than the value of the expression. If an out of range value is entered, the "Out of Range Value Entered" event will be generated.
Maximum	The maximum value permitted. If an expression is entered, the value entered by the operator must be not be more than the value of the expression. If an out of range value is entered, the "Out of Range Value Entered" event will be generated.
Step	The step used for the Raise and Lower keys. This field is used to control how much the value will change by when the Raise and Lower keys are used for data entry. Leaving the field at its default value will produce a step of one, while entering a zero value will effectively disable these keys.
Default	The default value for the field. If an expression is entered, the field will be loaded with the value of that expression before data entry is commenced. If the property is left at its default value, the field will contain whatever was previously held in the expression used to define its "Value" property.
Enable	Whether or not data entry is permitted. If an expression is entered for this property, it must evaluate to a non-zero value if data entry is to be allowed. If the expression is zero, the field will operate as if the "Mode" setting had been set to "read-only". The field is typically used for user access control.
Access Level	Used with the System Security feature to control user access.
Touch Entry	See touch units in Section E.

## Animation Default Control...

The Animation Defaults allows the user to determine the default template that will automatically appear when inserting a signed or unsigned 16-bit or 32-bit integer. Click on Tools from the main menu. Then choose Options and then Animation Defaults (see below).



As seen, the default templates can be set, as well as, the default min and max. Here a 16-bit signed integer will show a negative sign if it is below zero and will have a range from -32768 to 32767.

Before you download the database, you need to set up CommBlock A. Minimize **Display Pages**, maximize **Contents**, and choose **CommBlocks**.

Device	Address	Data Type	Size	Access	Update	Enable
A	INTERNAL	16-bit Signed	10	Read	Auto	Default
B	None	16-bit Signed	0	Read	Auto	Default
C	None	16-bit Signed	0	Read	Auto	Default
D	None	16-bit Signed	0	Read	Auto	Default
E	None	16-bit Signed	0	Read	Auto	Default
F	None	16-bit Signed	0	Read	Auto	Default
G	None	16-bit Signed	0	Read	Auto	Default
H	None	16-bit Signed	0	Read	Auto	Default
I	None	16-bit Signed	0	Read	Auto	Default
J	None	16-bit Signed	0	Read	Auto	Default
K	None	16-bit Signed	0	Read	Auto	Default
L	None	16-bit Signed	0	Read	Auto	Default
M	None	16-bit Signed	0	Read	Auto	Default
N	None	16-bit Signed	0	Read	Auto	Default
O	None	16-bit Signed	0	Read	Auto	Default
P	None	16-bit Signed	0	Read	Auto	Default
Q	None	16-bit Signed	0	Read	Auto	Default
R	None	16-bit Signed	0	Read	Auto	Default
S	None	16-bit Signed	0	Read	Auto	Default
T	None	16-bit Signed	0	Read	Auto	Default
U	None	16-bit Signed	0	Read	Auto	Default
V	None	16-bit Signed	0	Read	Auto	Default
W	None	16-bit Signed	0	Read	Auto	Default
X	None	16-bit Signed	0	Read	Auto	Default

By entering an arbitrary value of size 10 for Comms Block A, this memory location is automatically set up for internal. In other words, memory locations A[0] to.....A[9] are designated as Internal memory locations that we can Read/Write data to. Hit **F9** or click the download icon from the toolbar to make these changes to your HMI.

When you use the raise, lower or numeric Entry keys, you can only enter numbers from 0 to 20 into this Integer field.

**Connecting Your HMI to a PLC:** At this point you may want to connect your HMI to a PLC to perform simple tasks like turning a bit on and off. For an example, turn to page 2 of Section D of this manual.

## Direct PLC References

Programmers can use Direct PLC References in lieu of the CommsBlock mapping feature of EDICT-97. The following is a step by step example for using Direct PLC references in a new database via the Expression Wizard.

Integer Value

General | Data Entry

Name	Value
Value	Default
Transform	None
Template	00000
Mode	Read-Only

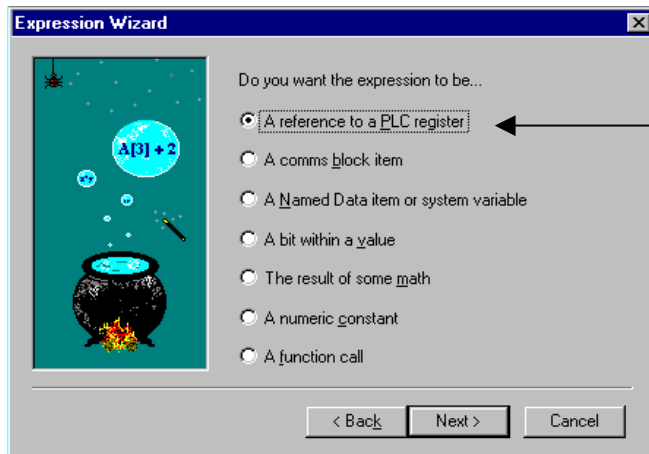
OK Cancel Help

Insert an Integer Value on a display page.

Click here to activate the Expression Wizard.



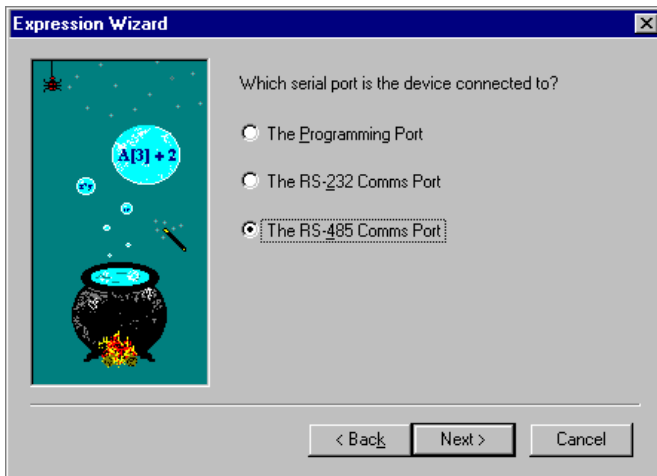
The Expression Wizard is started.



The Expression Wizard allows the programmer to choose Direct PLC references.



The Expression Wizard helps the programmer select the type of device.....



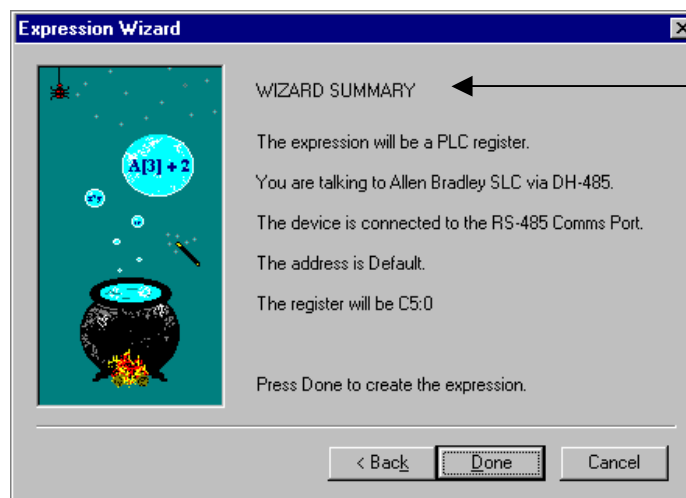
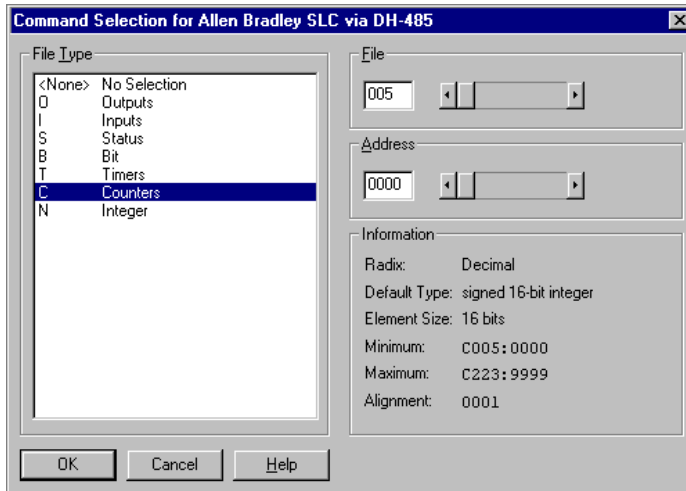
Prompts the choice of port on the panel that will be used to communicate with the target device....



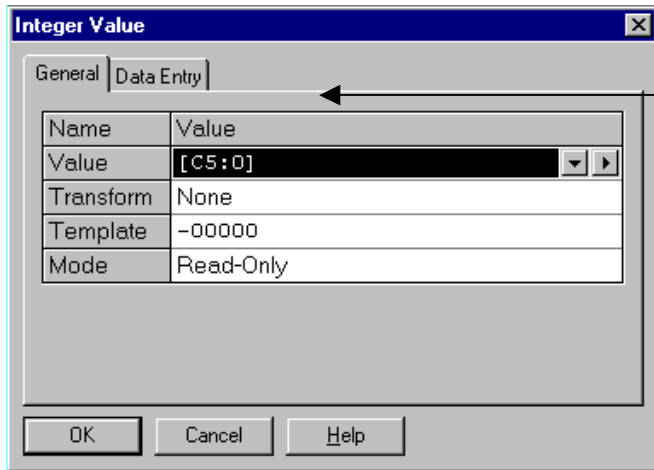
Prompts for the address of the target device....



Prompts for the register to be accessed in the target device....

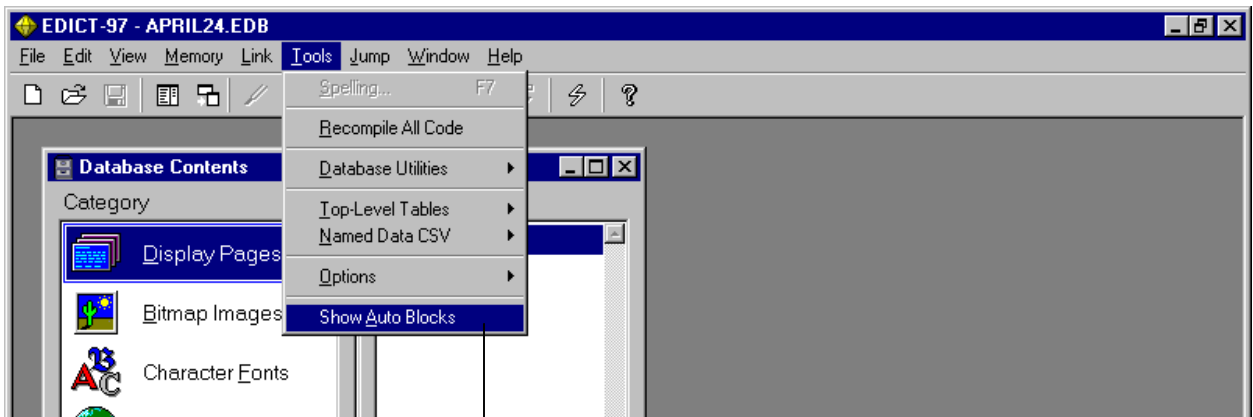


The Wizard Summarizes the choices made during the Expression Wizard.



Inserted Integer on display page after the programmer uses the Expression Wizard.

To view the Direct PLC references select **Show Auto Blocks** from the **Tools** Menu.



Automatic Block Table

	Device	Address	Data Type	Size
1	P03	C5:0	16-bit Signed	1
2	None	None	16-bit Signed	0
3	None	None	16-bit Signed	0
4	None	None	16-bit Signed	0
5	None	None	16-bit Signed	0
6	None	None	16-bit Signed	0
7	None	None	16-bit Signed	0
8	None	None	16-bit Signed	0
9	None	None	16-bit Signed	0
10	None	None	16-bit Signed	0
11	None	None	16-bit Signed	0
12	None	None	16-bit Signed	0

At the bottom are buttons for 'OK' and 'Help'.

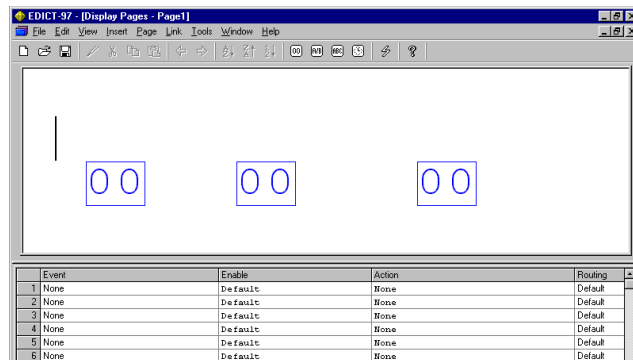
## Example 2: Inserting Multiple Integer Animation Items

### Return to Page1

Position your blinking cursor on Page1 so you can add another Integer value. Repeat steps 1 and 2 above (use A[2] instead of A[1]). Reposition your blinking cursor and repeat steps 1 and 2 again (use A[3] instead of A[2]). Your display page should look like this...

Hit **F9** or choose the download icon from the toolbar to update your HMI.

Use the numeric keys to change display values and hit the **Enter** key to move to the next field. The **Prev** and **Next** keys on your HMI move you from one data entry field to another. There are other modes to enter data and they are accessed by choosing **Page**, then **Properties**, and then



**Data Entry.** The following options are available when you choose **Page /Properties**.

## Display Page Properties

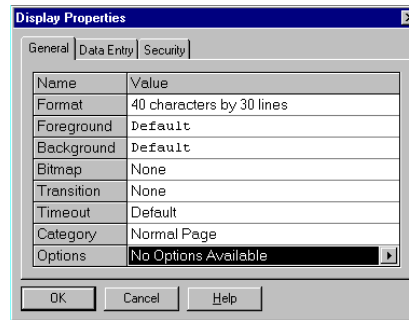
This dialog box is used to edit the properties of the current display page. Some of these properties can be left as "Default", in which case the default value defined by editing the properties (under Edit on the Menu line) of the Display Pages icon in the database contents window will apply.

### General Properties

The table below lists the general properties and their effects...

Property	Description
Format	The format of the display page. If the currently selected terminal supports more than one page format, the drop-down list can be used to select the format for this page. You will be warned if changing the format results in deleting animation items.
Foreground	Refers to graphic units (See Section E).
Background	Refers to graphic units (See Section E).
Transitions	Refers to graphic units (See Section E).
Bitmap	The background bitmap for the page. This option is used to select the background bitmap for terminals, which support graphical page formats. (See Section E).
Timeout	The page timeout in seconds. This value specifies the number of seconds without a key press after which the keyboard timeout event will be processed. The event will not actually cause anything to happen unless an event map states otherwise.
Category	The page category. This value is used to indicate that a page should perform a special system function. If a non-default value is selected, the system will change the default event handling for that page.

From the Menu choose **Page**, then **Properties**.



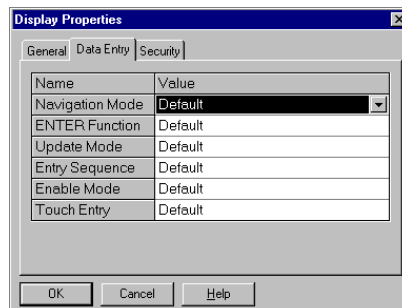
## Display Page Categories

The “Category” property of a display page is used to indicate what type of page should be displayed. The default setting produces an empty page upon which you can place your own animation items, and with very basic default event handling. Other settings cause the system to take over drawing the page and to provide more complex default event processing. The effect of this is to implement special system pages, such as those used to view active alarms, to set the real-time clock or to view the system status.

Category	Description
Alarm Viewer	Lists the currently active alarms, and lets you accept them.
Set Clock	Displays the real-time clock, and lets you modify its setting.
System Status	Display technical internal system status information
Event Viewer	Displays the contents of an Event Log.
Trend Viewer	Displays a graphical trend. See Section E
System Login	System Login page.
Touch Calibration	Refers to touch sensitive HMI's. Touch calibration page.
Touch Cleaning	Refers to touch sensitive HMI's. Touch cleaning page.

You can customize the exact behavior of one of these system pages by placing entries in the local event map of the page. If you want the default system behavior to be skipped for a given event, you can use the “Routing” column to indicate that the system should not be allowed to process the event once your action has been executed. Be careful when overriding system event handling, or you may find that the page no longer behaves as you would expect.

The current version of EDICT-97 does not allow you to modify what is displayed upon a system page, and any text or animation items entered via the display editor will be ignored and replaced with the system’s own text. Later versions will allow you to modify the system’s output and to add extra information of your own. For more information on the availability of this feature, please contact your supplier.



## Data Entry Properties

The table below lists the data entry properties and their effects...

Property	Description
Navigation Mode	How the cursor keys should behave. If "Linear" is selected, the keys will move the cursor from one end of the list to the other, stopping at each end. "Cyclic" will cause the cursor to wrap-around, suppressing some events in the process.
ENTER Function	How the Enter key should behave. If "Move" is selected, the key will save the data and move on to the next field. Selecting "Commit" will simply save the data, and leave the cursor on the current field until a cursor key is used to move it.
Update Mode	How the display will be updated. If "Limited" is selected, other animation fields will only be updated when the entry field changes or a new field is selected. "Normal" indicates that the display should also update after each Comms scan.
Entry Sequence	How the entry order is calculated. This field is used to select how EDICT-97 sorts data entry fields when the fields are arranged in a grid pattern. You can indicate that you wish to scan the data by rows or by columns, as required by your application.
Enable	Whether or not data entry is permitted. If an expression is entered for this property, it must evaluate to a non-zero value if data entry is to be allowed. If the expression is zero, the field will operate as if the "Mode" setting had been set to "read-only". The field is typically used for user access control.
Touch Entry	Refers to touch sensitive HMIs. See Section E.

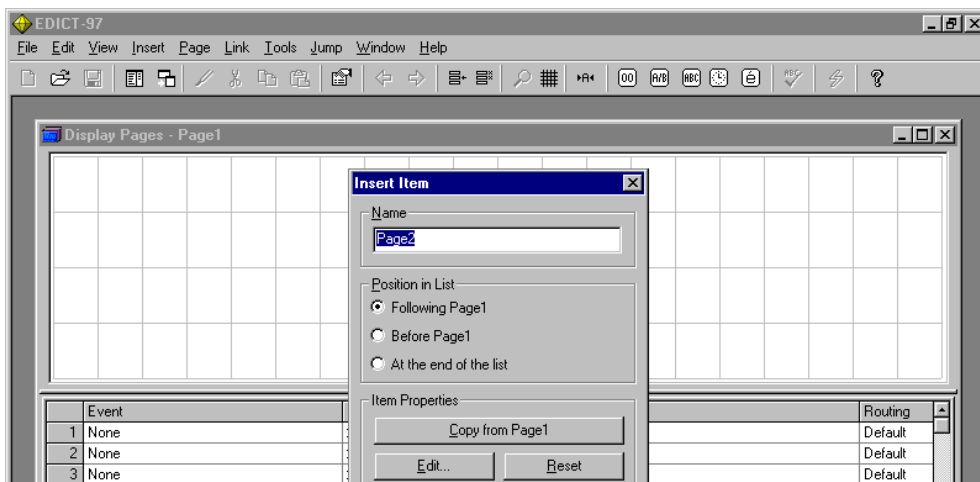
Note that selecting a "Normal" display update may cause the cursor to flicker somewhat on some terminals, as the display driver has to turn off the cursor when the other fields are being updated. You should not select this setting unless you need live data on a data entry page.

## Default Page Properties

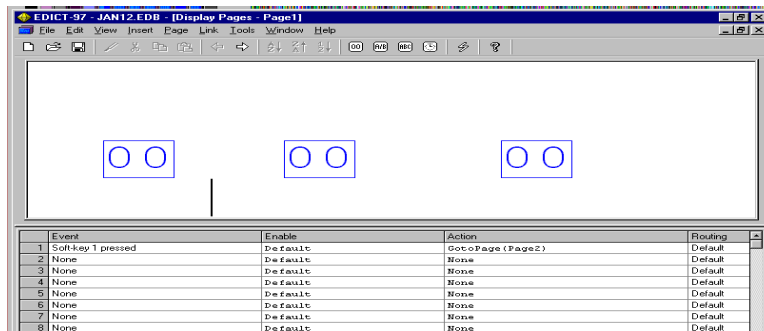
This dialog box is used to edit the default properties of all display pages. Each value can be overridden using the properties dialog box for the page in question. If you need to set a property for most of your pages, you should do so using this dialog box and then modify the default values on the pages where other settings are required. To access this Global set up box, go to the Database Contents page and choose **Edit**, then **Properties**.


## Example 3: Using Events and Actions(Functions)

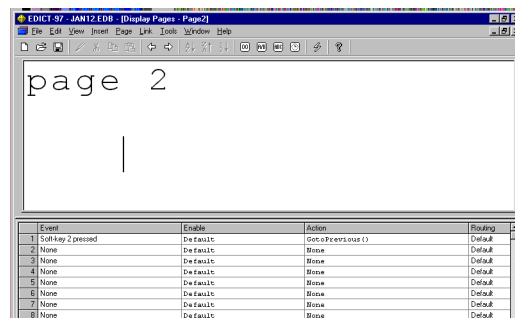
Return to **Page1** and choose **Page**, then **Insert** and click OK to Page2.



Now that you have 2 Pages, you need a way to move from 1 page to another. You will remember from the Panel diagram on Page 1 of this manual that your HMI has a combination of Soft and Function Keys. In this example SoftKeys 1 and 2 will be used for this purpose. Page1 should look like ..



Notice that midway down the page **Soft-key 1 pressed** is entered under Event and **GotoPage(Page2)** is entered for Action. Now hit  on your toolbar to go to page 2.



Notice that **Soft-key 2 pressed** is entered for Event 1 and **GotoPrevious( )** is entered for Action. You should also enter some text to distinguish which page you are viewing on your HMI. Hit **F9** or hit the download icon on your toolbar, Now Softkey 1 and 2 on your HMI should toggle you between the two pages. This example introduces the use of **Events** and **Action (Functions)**. Simply stated **Events** are things that happen (like a SoftKey being pressed) and **Actions** are things that are programmed to happen after a specific **Event** happens. The following Event Index and Function Index are also available as drop down Windows during programming.

## Event Handling

The event handling system is the key to how EDICT-97 operates. It is used to define what should happen when a given event occurs. Events are generated whenever the operator presses a key on the operator terminal, or in response to certain internal changes within EDICT-97. By assigning actions to events, you can modify just about any aspect of the terminal's behavior in a very flexible way.

The relationship between events and actions is defined using event maps. Each event map lists the events to be processed, and the actions to be performed in response to those events. The global event map defines how events should be processed, no matter which page is currently displayed. Each page also has its own event map, used to define how events should be processed when that page is selected. This page-specific event map can be used to override global event map entries if required.

## Events Index

The following table lists the events recognized by EDICT-97, the associated abbreviation, and any associated value held in the "Param" system variable. Note that this variable contains the parameter only during the processing of the event in question, and so you will not be able to display it on a page using an animation item. If you want to examine the parameter in that way, store it in a variable in response to the event, and display that variable.

Event	Abbreviation	Parameter
Soft-Key Pressed	SK1, SK2...	Undefined.
Function Key Pressed	F1, F2...	Undefined.
Exit Key Pressed	EXIT	Undefined.
Next Key Pressed	NEXT	Undefined.
Prev Key Pressed	PREV	Undefined.
Raise Key Pressed	RAISE	Undefined.
Lower Key Pressed	LOWER	Undefined.
Enter Key Pressed	ENTER	Undefined.
Delete Key Pressed	DELETE	Undefined.
Menu Key Pressed	MENU	Undefined.
Mute Key Pressed	MUTE	Undefined.
Alarms Key Pressed	ALARMS	Undefined.
Numeric Key Pressed	NUM	Value of Key.
Touch Screen Pressed	TOUCH	Undefined.
Early Initialization (Before comms are started)	EARLY	Undefined.
Keyboard Timeout	TIMEOUT	Timeout in Seconds.
System Initialized	INIT	Undefined.
Page Selected	SEL	Undefined.
Page Removed	REM	Undefined.
Page Update Start	PAINT1	Undefined.
Page Update End	PAINT2	Undefined.
Out Of Range Value	RANGE	Undefined.
One Second Tick	ONE	Undefined.
Comms Update	UPDATE	Undefined.
Comms Error	ERROR	Comms Error Flags.
New Field Selected	FIELD	Entry Field Index.
Tab Past First Field	FIRST	Undefined.
Tab Past Last Field	LAST	Undefined.
All Input Complete	ALL	Undefined.
Any Key Pressed	ANY	Keyboard Code.
User Generated Event	USER	User Parameter.

Alarm Activate	ALARM	Alarm Index.
Alarm Accepted	ACCEPT	Alarm Index.
Alarm Cleared	CLEAR	Alarm Index.
Incoming Connection	INCOMING	Port Number.
New Link Established	CONNECT	Port Number.
Existing Link Broken	BROKEN	Port Number.
Login Successful	LOGIN	Undefined.
Login Failed	FAILED	Undefined.
User Logged Off	LOGOFF	Undefined.
User Timeout	USERT	Timeout in Seconds.
Page Access Failed	ACCESS	Undefined.

## Notes

1. "Key Released" codes are formed from the base abbreviation plus the letter "R".
2. "Auto Repeat" codes are formed from the base abbreviation plus the letter "A".
3. Where the parameter is "Undefined", you should not assume a value of zero.

## The Action Builder

The Action Builder dialog box is accessed by clicking on the right-pointing arrow within an action field, or by using the Shift+F2 key combination. The dialog box contains a list of functions, which may be used within an action, together with a section to assist you in defining the arguments of the function you have selected.

You should select the required function from the list on the left-hand side of the dialog box, and examine the "Syntax" section to see what arguments are required. The three boxes on the right-hand side can then be used to enter the arguments. If the argument is the name of some item within the database, EDICT-97 will fill the associated drop-down list with the names of the items of the correct type. Entering the name, which does not exist, will produce a prompt as to whether or not a corresponding item should be created.

The "Details" button can be used to display a help page concerning the currently selected function. This page will explain what the function does, what arguments it takes, and give a simple example of the function in use. It will also describe any things to watch out for when using the function, and provide links to related functions.

## Function Index (See Section A for further details on each Function)

### Active Functions

The table below lists functions which can be used within actions...

<a href="#"><u>AcceptAlarm</u></a>	<a href="#"><u>AcceptAll</u></a>	<a href="#"><u>Beep</u></a>	<a href="#"><u>ClearRx</u></a>
<a href="#"><u>ClearTx</u></a>	<a href="#"><u>CopyData</u></a>	<a href="#"><u>Dispatch</u></a>	<a href="#"><u>Fill</u></a>
<a href="#"><u>FlipEntrySign</u></a>	<a href="#"><u>GotoPage</u></a>	<a href="#"><u>GotoPrevious</u></a>	<a href="#"><u>HoldTx</u></a>
<a href="#"><u>MuteSiren</u></a>	<a href="#"><u>PostEvent</u></a>	<a href="#"><u>PrintFormFeed</u></a>	<a href="#"><u>PrintMessage</u></a>
<a href="#"><u>PrintNewLine</u></a>	<a href="#"><u>PrintReport</u></a>	<a href="#"><u>PrintString</u></a>	<a href="#"><u>PrintTimeStamp</u></a>

<a href="#"><u>ReadBlock</u></a>	<a href="#"><u>Run</u></a>	<a href="#"><u>SerialPrint</u></a>	<a href="#"><u>SerialRead</u></a>
<a href="#"><u>SerialWrite</u></a>	<a href="#"><u>SetBreak</u></a>	<a href="#"><u>SetCommsTask</u></a>	<a href="#"><u>SetRTS</u></a>
<a href="#"><u>SetTimer</u></a>	<a href="#"><u>SirenOn</u></a>	<a href="#"><u>Sleep</u></a>	<a href="#"><u>StopSystem</u></a>
<a href="#"><u>TriggerAlarm</u></a>	<a href="#"><u>UseAutoEnables</u></a>	<a href="#"><u>UseHalfDuplex</u></a>	<a href="#"><u>WriteBlock</u></a>

### Passive Functions

The table below lists functions which can be used within expressions...

<a href="#"><u>Abs</u></a>	<a href="#"><u>CallFloat</u></a>	<a href="#"><u>CallInt</u></a>	<a href="#"><u>CallLong</u></a>
<a href="#"><u>CallString</u></a>	<a href="#"><u>Format</u></a>	<a href="#"><u>GetTimer</u></a>	<a href="#"><u>IsOnLine</u></a>
<a href="#"><u>Left</u></a>	<a href="#"><u>Len</u></a>	<a href="#"><u>HexVal</u></a>	<a href="#"><u>MakeChar</u></a>
<a href="#"><u>Max</u></a>	<a href="#"><u>Mean</u></a>	<a href="#"><u>Mid</u></a>	<a href="#"><u>Min</u></a>
<a href="#"><u>PopDev</u></a>	<a href="#"><u>Random</u></a>	<a href="#"><u>Right</u></a>	<a href="#"><u>Sgn</u></a>
<a href="#"><u>Sqrt</u></a>	<a href="#"><u>StdDev</u></a>	<a href="#"><u>Val</u></a>	

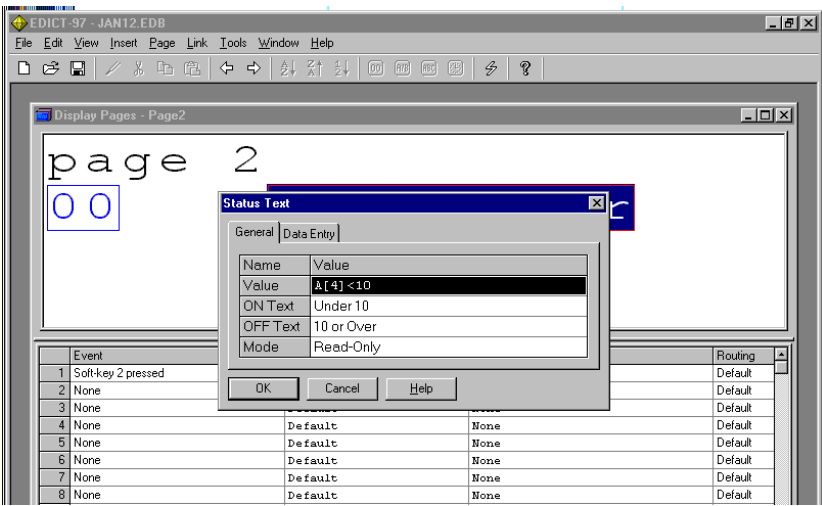
## Inserting Animation Items

### Animation Items

Animation items are used to put live data on to a display page or a printed report. They are placed on the page by using the various options under the "Insert" menu. The table below lists the various animation items that EDICT-97 supports, together with an indication as to which support data entry when used on display pages...

<b>Name</b>	<b>Description</b>	<b>Data Entry</b>
<a href="#"><u>Integer Value</u></a>	Displays an integer value.	Yes.
<a href="#"><u>Real Number</u></a>	Displays a floating point value.	No.
<a href="#"><u>Status Text</u></a>	Displays one of two strings.	Yes.
<a href="#"><u>Quad Text</u></a>	Displays one of four strings.	Yes.
<a href="#"><u>Message Text</u></a>	Displays a string from a numbered list.	Yes.
<a href="#"><u>Decode Text</u></a>	Displays a string from a condition list.	No.
<a href="#"><u>General Text</u></a>	Displays a general string expression.	No.
<a href="#"><u>Bar Graph</u></a>	Displays a horizontal bar graph.	Yes.
<a href="#"><u>Indicator</u></a>	Displays a on-off graphics indicator	Yes.
<a href="#"><u>Time / Date</u></a>	Displays the current time and/or date.	No.

Return to Page 2 in your EDICT-97 file. On line 2 Insert Integer from your **Toolbar** ( Make it: Value A[4], Template 00, Mode Data Entry) and after several spaces, choose **Insert Status Text** from your **Toolbar** ( or **Insert/Text/Status** from your Menu)



Use your cursor and make the above changes to the **Insert Status Text** Window. The text on your HMI will be “Under 10” when the Integer value A[4] is under 10, and the text will be “ 10 or Over “ when the Integer value of A[4] is 10 or over. Click **OK** and download to your HMI.

**Status Text Animation Item**

The animation item is used to display one of a pair of text strings, based upon the logical value of an expression. For example, it can be used to display the status of a single bit, or to display a string based upon whether a numeric value is over a certain limit. This field can be used for display, and also for data entry. Data entry is not available when working with printed reports. The table below lists the properties of this animation item...

Property	Description
Value	An integer expression used to select the text to be displayed. If the field is set to “read-only” operation, any expression can be used. If “Data Entry” mode is selected, a writable value must be entered. Note that only the truth or otherwise of the expression is considered, with any non-zero value being considered <i>true</i> .
ON Text	The text to display when “Value” is <i>true</i> .
OFF Text	The text to display when “Value” is <i>false</i> .
Mode	Whether or not data entry is to be enabled. If this property is set to “read-only”, the animation item will simply display the string selected by the “Value” property. Selecting “Data Entry” will allow the operator to change the value of the field, using the Raise and Lower buttons.

In addition to Status Text Animation Items there are also: Quad Text Animation Items, Message Text Animation Items, DecodeText Animation Items, General Text Animation Items, Horizontal Bar Animation Items, Indicator Animation Items and Time and Date Animation Items.

### Quad Text Animation Item

The animation item is used to display one of four text strings, based upon the value of the bottom two bits of an expression. It is often used to display the status of a three way valve, based upon the limit switches at either end of its travel. This field can be used for display, and also for data entry. Data entry is not available when working with printed reports.

The table below lists the properties of this animation item...

Property	Description
Value	An integer expression used to select the text to be displayed. If the field is set to "read-only" operation, any expression can be used and the bottom two bits will be considered when selecting the string to be displayed. If "Data Entry" mode is selected, a writeable value must be entered and a value of from 0 to 3 will be written to the expression, with the other bits being set to zero.
Text 0	The text to display when the bottom bits of "Value" are 00.
Text 1	The text to display when the bottom bits of "Value" are 01.
Text 2	The text to display when the bottom bits of "Value" are 10.
Text 3	The text to display when the bottom bits of "Value" are 11.
Mode	Whether or not data entry is to be enabled. If this property is set to "read-only", the animation item will simply display the string selected by the "Value" property. Selecting "Data Entry" will allow the operator to change the value of the field, using the Raise and Lower buttons.

### Message Text Animation Item

This animation item is used to display a text string chosen from the message table, with the selection being based upon the value of an expression. You can select from either the global message table, or a table local to a given animation item. This field can be used for display, and also for data entry. Data entry is not available when working with printed reports.

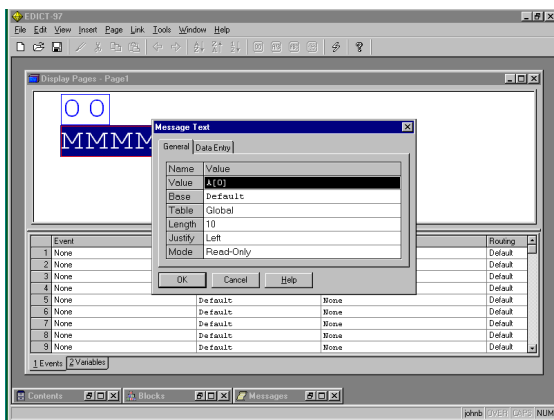
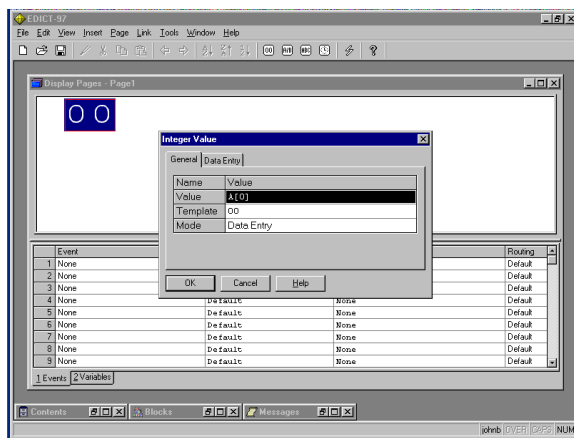
The table below lists the properties of this animation item...

Property	Description
Value	An integer expression used to define the number of the message to be displayed. If the field is set to "read-only" operation, any expression can be used. If "Data Entry" mode is selected, a writable value must be entered.
Base	The message to show when "Value" evaluates to zero. Leaving this property at "Default" will result in an invalid message being selected in such circumstances, with the valid messages starting when "Value" is one. This field is used to introduce an offset into the message selection process, without having to make "Value" into a non-writable value by including the offset within that expression. It is rarely used with a local message table.
Table	The message table to used. If you leave this property set to "Global", the message will be selected from the global message table. Expanding the field allows you to create a message table local to this animation field. This can make life easier sometimes, but can be wasteful of memory and harder to maintain if the same table is to be repeated many times.

Length	The length of the field on the display page. You should enter a value from 1 to the maximum number of characters permitted by the display page format and the cursor position. Messages longer than this value will be clipped according to the “Justify” setting.
Justify	How the message is to be formatted. This setting controls how EDICT-97 pads-out messages which are shorter than the “Length” setting, and which characters EDICT-97 discards if a message is longer than the animation field. It also controls where the cursor appears if the field is selected for data entry.
Mode	Whether or not data entry is to be enabled. If this property is set to “read-only”, the animation item will simply display the message selected by the “Value” property. Selecting “Data Entry” will allow the operator to change the value of the field, using the Raise and Lower buttons.

---

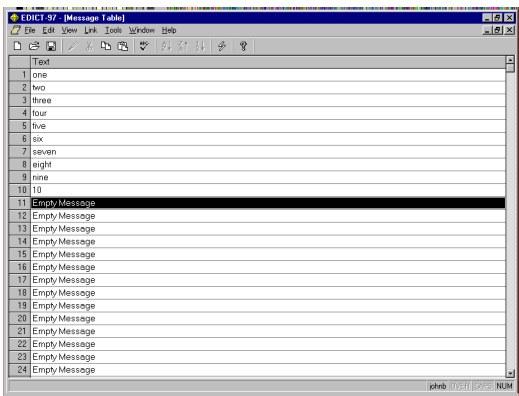
### Example of Inserting Message Text Animation:



First Insert **Integer Value** onto your page and designate it data entry.

Then Insert **Text /Message** and designate the Message number value equal to the value of the Integer field A[0].

Finally make the following entries into the Message Table.



Hit **F9** to download and you will see the message on your display page change based upon the value of A[0].

**Decode Text Animation Item**

This animation item is used to select a string from a table, based upon the *true* or *false* value of an expression associated with each string. EDICT-97 scans the table, and selects the first string for which the controlling expression is *true*. This allows complex decoding functions to be performed, including such things as bit-level prioritization of messages, or multiple decodes of numeric values.

The table below lists the properties of this animation item...

Property	Description
Table	The decode table of this field. You should expand the field by pressing the F2 key, and then enter pairs of controlling expressions and strings. EDICT-97 will display the first string for which the controller expression is <i>true</i> , or left empty. The order of the entries in the table is obviously very important.
Length	The length of the field on the display page. You should enter a value from 1 to the maximum number of characters permitted by the display page format and the cursor position. Strings longer than this value will be clipped according to the “Justify” setting.
Justify	How the string is to be formatted. This setting controls how EDICT-97 pads-out strings which are shorter than the “Length” setting, and which characters EDICT-97 discards if a string is longer than the animation field.

**General Text Animation Item**

This animation item displays the value of a string expression. As well as displaying string data from the PLC or other Comms devices, it can be used together with the “CallString” function to implement custom animation types. For example, you might run a program which examines level data, and then either returns the value formatted using the “Format” function, or an indication that the level is too high or low.

The table below lists the properties of this animation item...

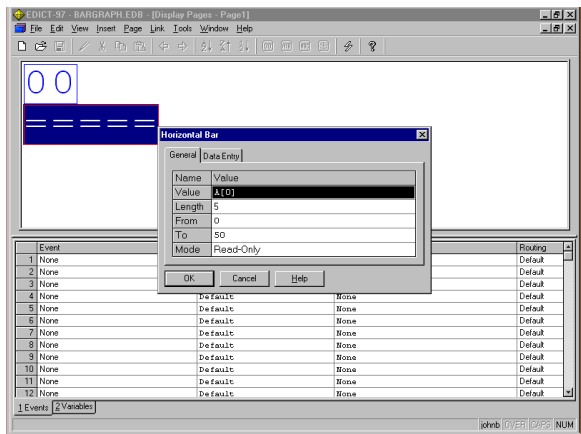
Property	Description
Value	A string expression giving the text to be displayed. If the field is set to "read-only" operation, any expression can be used. If "Data Entry" mode is selected, a writable value must be entered.
Length	The length of the field on the display page. You should enter a value from 1 to the maximum number of characters permitted by the display page format and the cursor position. Strings longer than this value will be clipped according to the "Justify" setting.
Justify	How the string is to be formatted. This setting controls how EDICT-97 pads-out strings which are shorter than the "Length" setting, and which characters EDICT-97 discards if a string is longer than the animation field.

### Horizontal Bar Animation Item

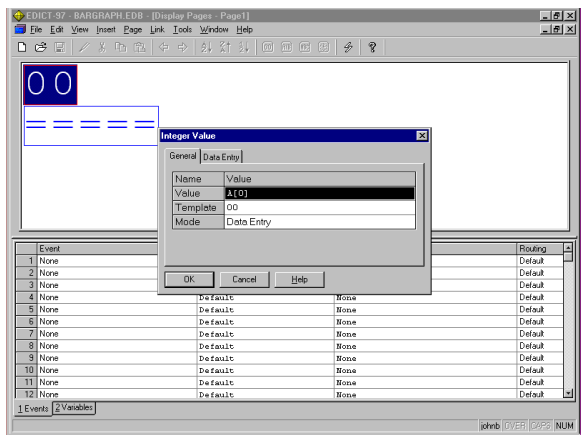
This animation item displays a numeric value as a horizontal bar graph. If it is used on a terminal with a graphics display, single-pixel resolution will be used. If it is used on a printed report or on a character-based terminal, character resolution will be used. This field can be used for display, and also for data entry. Data entry is obviously not available when working with printed reports. The table below lists the properties of this animation item...

Property	Description
Value	An integer expression used to define the value to be displayed. If the field is set to "read-only" operation, any expression can be used. If "Data Entry" mode is selected, a writable value must be entered.
Length	The length of the field on the display page. You should enter a value from 1 to the maximum number of characters permitted by the display page format and the cursor position. This field will partly determine the bar graph scaling.
From	An integer expression representing the value to be represented by the bar graph when no pixels are shaded. This value should always be less than that entered for the "To" property or undefined results will be produced.
To	An integer expression representing the value to be represented by the bar graph when all pixels are shaded. This value should always be greater than that entered for the "From" property, or undefined results will be produced.
Mode	Whether or not data entry is to be enabled. If this property is set to "read-only", the animation item will simply display the message selected by the "Value" property. Selecting "Data Entry" will allow the operator to change the value of the field, using the Raise and Lower buttons.

**Bar Graph Animation Example:** An application requires that an integer value of 0 to 50 be displayed with a corresponding 5-segment bar graph display.



Use A[0] as the source value ..  
And insert the horizontal bar graph below. Hit **F9** to download. Note when you enter values from 0 to 50 for A[0] that the bar graph changes at the midway point(i.e.first bar graph segment appears at A[0]=5 and the second bar graph appears at A[0]= 15).



### Indicator Animation Item

This animation item is used to represent a digital value. It occupies a single character on the display or printed report, with the character being shaded-in if the controlling value is *true*, or left empty if the value is *false*. Terminals with graphic displays may choose to render this animation item using a filled or empty box, although this feature may only operate in some display modes. This field can be used for display, and also for data entry. Data entry is obviously not available when working with printed reports.

The table below lists the properties of this animation item...

Property	Description
Value	An integer expression used to define the value to be displayed. If the field is set to “read-only” operation, any expression can be used. If “Data Entry” mode is selected, a writable value must be entered. A non-zero value will “light up” the indicator, while a zero value will leave it in its default state.

Mode	Whether or not data entry is to be enabled. If this property is set to "read-only", the animation item will simply display the message selected by the "Value" property. Selecting "Data Entry" will allow the operator to change the value of the field, using the Raise and Lower buttons.
------	--

---

### Time and Date Animation Item

This animation item is used to display the current time or date. It has a single "Template" property, which is used to show how the time or date should be formatted. Each character in the template either represents a character to be literally copied to the output stream, or a placeholder for time or date information.

The table below lists the possible placeholder characters, and explains their effects...

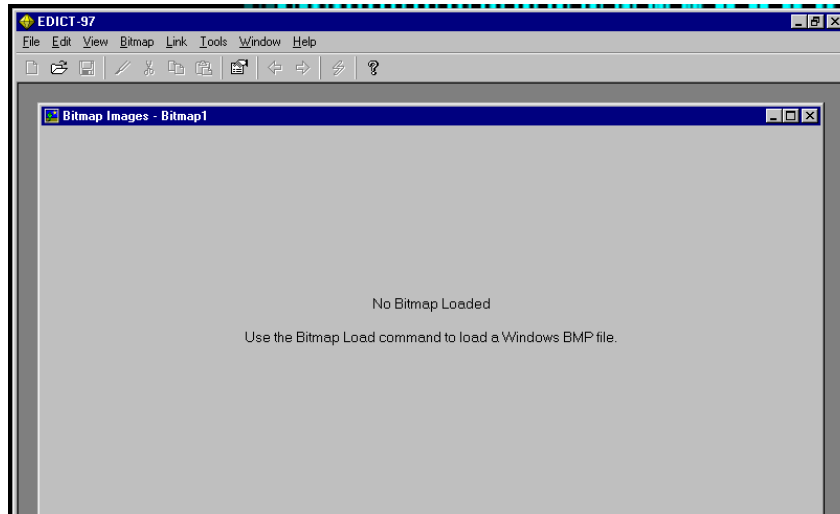
Character	Description
'h'	Will be replaced with the next available digit of the current hour.
'm'	Will be replaced with the next available digit of the current minute.
's'	Will be replaced with the next available digit of the current second.
'Y'	Will be replaced with the next available digit of the current year.
'M'	Will be replaced with the next available digit of the current month.
'D'	Will be replaced with the next available digit of the current date.

Note that the time is always shown in 24-hour or military format.

## Section 2 Bitmaps

N/A to Text Based Units

(Graphic Units Only)



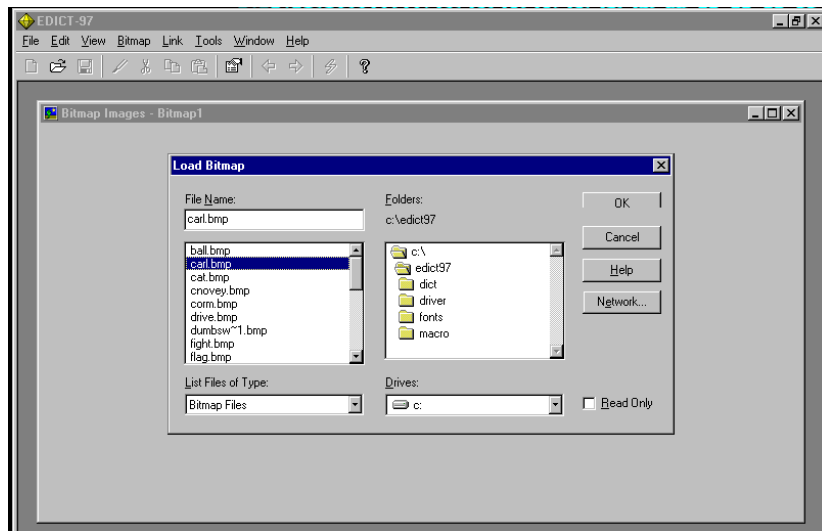
### Bitmap Images

Bitmap Images EDICT-97's bitmap images are used to store pre-drawn bitmaps to be displayed by operator terminals with a graphics display. Bitmaps may be used in two ways: Bitmaps equal in size to the display may be selected as backdrops for particular pages, or smaller bitmaps may be used as animation items. When a bitmap is used as an animation item, it can be divided into a number of sub-images, with the choice of which sub-image to display being made on the basis of data from the PLC or other remote device.

### The Load Bitmap Dialog Box

This dialog box is used to select a bitmap file to be loaded.

How To	Description
Specify a File	Select the name from the left-hand list, or type it into the edit box above the list. Filenames must be limited to a maximum of eight characters in length.
Change Directory	Select the new directory from the right-hand list, or type the path into the edit box and press <b>Enter</b> . Remember that Windows 95 style long filenames are not supported.
Change Drive	Select a new drive from the drop-down list below the right-hand directory list. Alternatively, type the drive letter and a colon into the edit box, and press the <b>Enter</b> key.



**Note: Imported Bitmaps must be monochrome or 16 color and less than 60k in size.  
The Width of an imported Bitmap must be an even number of pixels.**

The Bitmap shown in the Window above is an example of a Full Width Image.



### Bitmap Editing

EDICT-97 does not contain any facilities to edit bitmaps. Rather, you should use your favorite bitmap editor to prepare the image as a BMP file, and then use the "Load" command on the "Bitmap" menu to load it into EDICT-97. Once you have done this, EDICT-97 will remember the source pathname, so that you can update EDICT-97's copy of the bitmap simply by using the "Update" command instead.

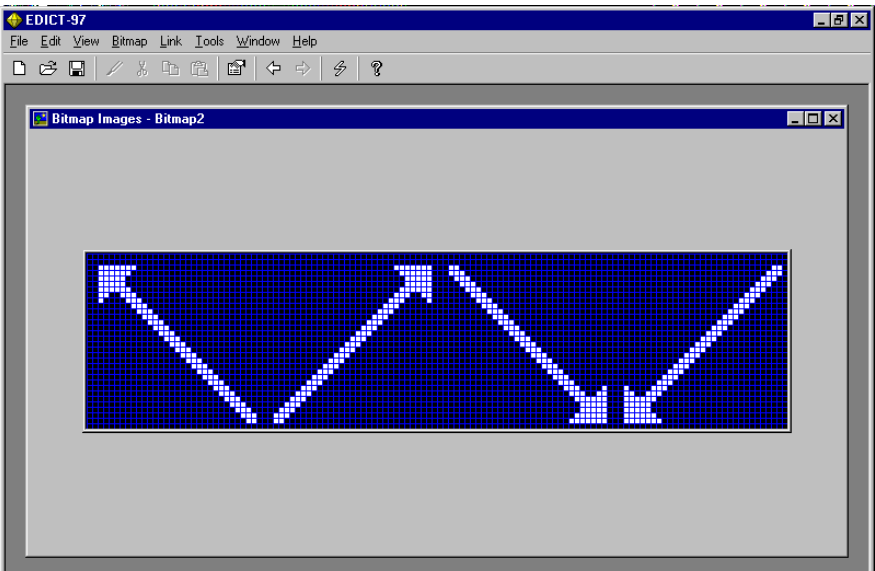
### View Menu Commands

The window provides the following commands on the "View" menu...

Command	Description
Maintain Aspect	If enabled, this option forces EDICT-97 to scale the bitmap such that the x and y scales are equal. This maintains the aspect ratio of the original bitmap; such that circles remain circular.
Show Grid	If enabled, EDICT-97 will show a pixel grid on top of the bitmap, providing that the scale is such that it can be displayed clearly. If the

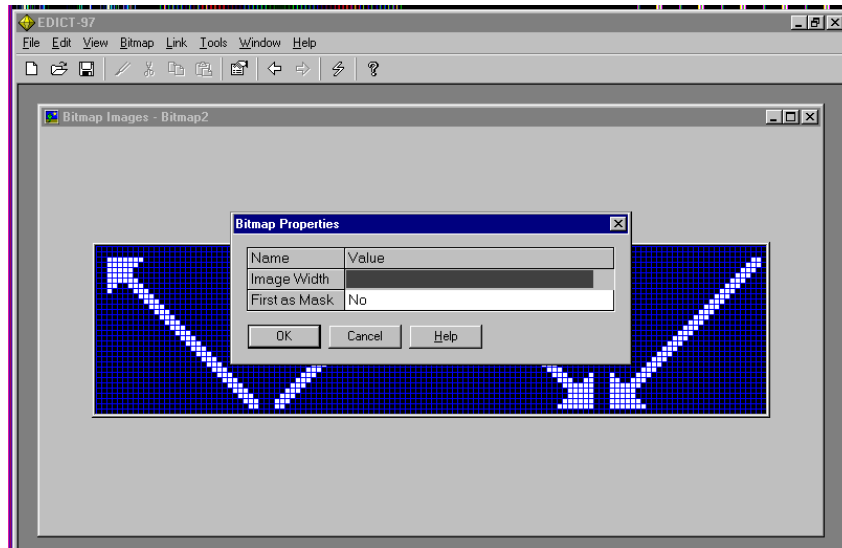
	bitmap has multiple sub-images, the grid will indicate where the sub-image boundaries occur.
View as Drawn	If enabled, EDICT-97 will show bitmaps with sub-images such that you can preview each sub-image in turn by using the “Next State” and “Previous State” commands. If this option is disabled, EDICT-97 will display the entire bitmap at once.

It is also possible to import Bitmaps with sub-images. This powerful feature allows for custom animation. The following example shows an imported bitmap, which contains 4 sub-images. Each sub-image is 32 pixels wide. The result will be an animated arrow rotating to four positions based upon the status of the Control Value.



Bitmap Properties properties of the current bitmap...

Property	Description
Image Width	The number of pixels for each sub-image. If you leave this property as “Whole Bitmap”, EDICT-97 will display the whole bitmap when it is selected as the source for a bitmap graphical animation item. If you define a pixel width, EDICT-97 will use a control value to choose one of a number of sub-images, each of which assumed to be the indicated number of pixels in width. A value of zero will show the first sub-image, value of one will show the second sub-image, and so on.
First as Mask	Whether to use the first sub-image as a mask. If this option is set to “Yes” and the bitmap is split into a number of sub-images, EDICT-97 will use the first sub-image as a mask to define which pixels on the display will be updated. Black pixels on the mask will be left as they were before the bitmap was drawn, while white pixels will be set to the value specified in the selected sub-image. This allows the use of non-rectangular images.



An image width of 32 pixels was chosen for this example because the sub-images were each 32 pixels wide.

## Section 3 Character Fonts

### N/A to Text Based Units

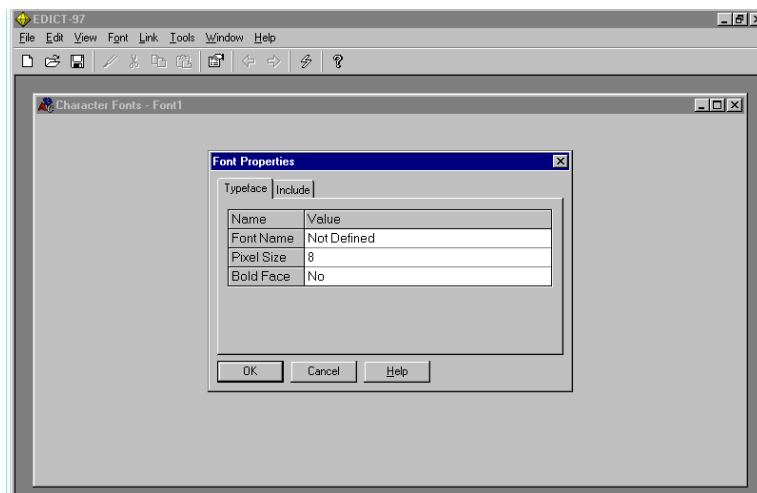
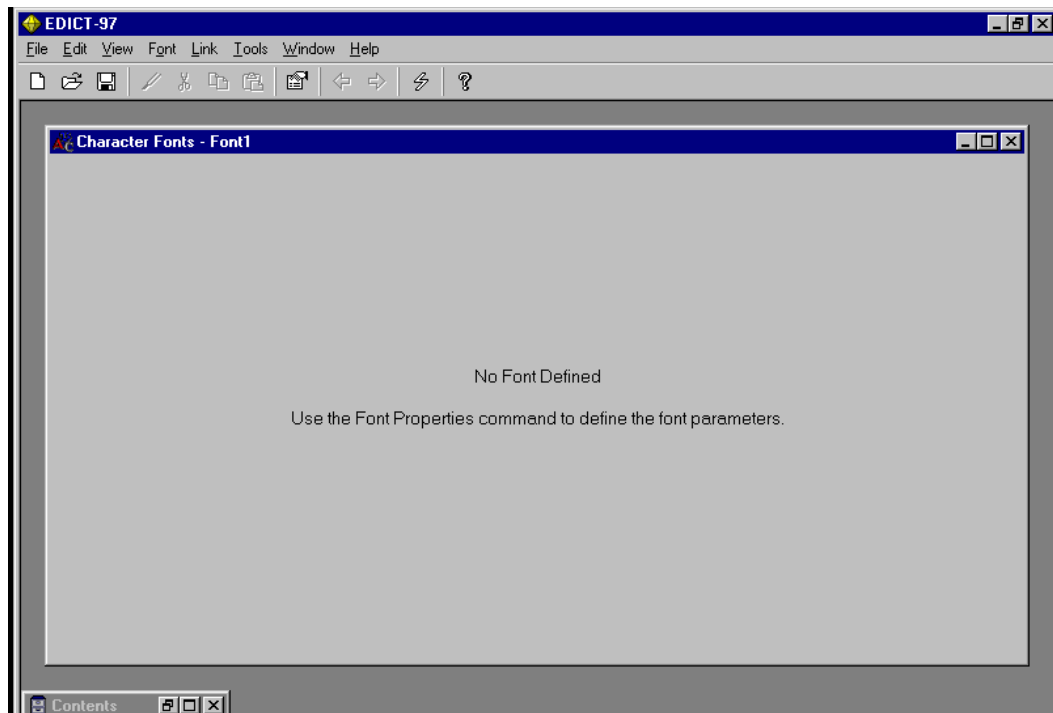
#### **The Character Fonts Window**

This window is used to define the downloadable Character Fonts to be used by your database in addition to the six “resident” fonts stored within the firmware of the operator terminal. Each downloaded font is a rendering of a given size of a TrueType or bitmap font installed on your system, which EDICT-97 will convert to a bitmap format suitable for use, by the terminal. To save memory, you can limit which characters on the font are downloaded, as many applications do not need the wide range of characters offered by most fonts by editing the font properties.

#### **IMPORTANT:**

You are reminded that TrueType fonts are licensed software. It is your responsibility to ensure that you have the permission of the copyright owner to convert it into a bitmap for downloading in this way before selecting it for use.

To select a Font **Choose Font/Properties.**



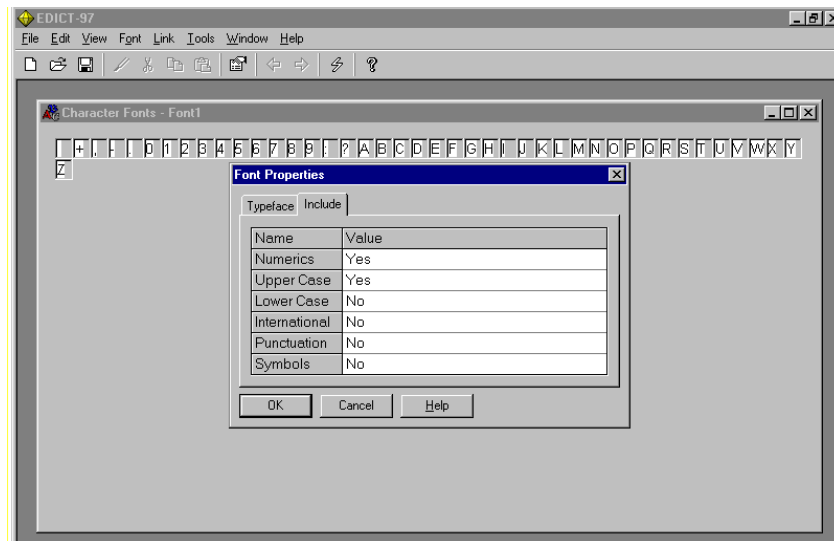
### **Character Font Properties**

This dialog box is used to edit the properties of the current font...

Property	Description
Font Name	The name of the font to be rendered. See the notice below.
Pixel Size	The height in pixels of the resulting font. Note that many fonts will not take up the whole of their character cell size, as they include some padding known as "leading". You may need to take this into account when specifying the pixel size.
Bold Face	Whether or not the font should be rendered in bold face.

There are also a number of properties used to define which characters will be included in the downloaded form of the font. The categories are hopefully self-explanatory, but please note that certain punctuation characters are always included if numeric or alphabetical characters are selected. This avoids having to download the entire punctuation set simply to include such things as decimal points and commas.

Once a Font has been selected you can then select which characters you will be using in your database. **Choose the Include Tab from the Font Properties Window.**



**Note: When selecting which items to include for a particular Font remember that each item (e.g. Numeric, Upper Case etc) uses memory in your Graphic Panel.**

## Section 4 Global Events

	Event	Enable	Action	Routing
1	None	Default	None	Default
2	None	Default	None	Default
3	None	Default	None	Default
4	None	Default	None	Default
5	None	Default	None	Default
6	None	Default	None	Default
7	None	Default	None	Default
8	None	Default	None	Default
9	None	Default	None	Default
10	None	Default	None	Default
11	None	Default	None	Default
12	None	Default	None	Default
13	None	Default	None	Default
14	None	Default	None	Default
15	None	Default	None	Default
16	None	Default	None	Default
17	None	Default	None	Default
18	None	Default	None	Default
19	None	Default	None	Default
20	None	Default	None	Default
21	None	Default	None	Default
22	None	Default	None	Default

### Event Map Properties

Each event map entry has the following properties...

Property	Description
Event	The event to be matched by this row. The field's drop-down list can be used to select an event, or you may enter the first few letters of the event's abbreviation directly from the keyboard. Turn to the Event Index section in Section A of this manual to see a full list of events, together with their associated abbreviations.
Enable	An enabling expression for this row. If you enter an expression in this field, the row will only be activated when the expression is <i>true</i> . This can be used to implement a simple form of "if-else" construction within the event map itself.
Action	The action to be performed. This field defines what will happen when the event is matched. You can either enter the code directly, or expand the cell and use the Action Builder to create the action for you. Turn to the Function Index in Section A of this manual to see a full list of Functions
Routing	Where to send the event next. This column controls what EDICT-97 will do with the event when it has been processed by this row. The default routing is from a local event map to the global event map, and thence to the default handler within EDICT-97. Using this property to specify the routing required can modify this behavior. Options: None, System, Global

	Event	Enable	Action	Routing
1	Soft-key 1 pressed	A[4]>10	PrintReport(0, Report1)	Default
2	F1 pressed	Default	GotoPrevious()	Default
3	Comms update	Default	CopyData(B[1], C[1], 5)	Default
4	None	Default	None	Default
5	None	Default	None	Default

### Examples:

1) When Soft-Key 1 on the HMI is pressed and the Numeric value of CommBlock A[4] is greater than 10; Report 1 will be printed.(For more detail on PrintReport Function look, in Function Index in section A of this manual.)

2) When Function Key 1 on the HMI is pressed, the HMI display will return to the previous page.

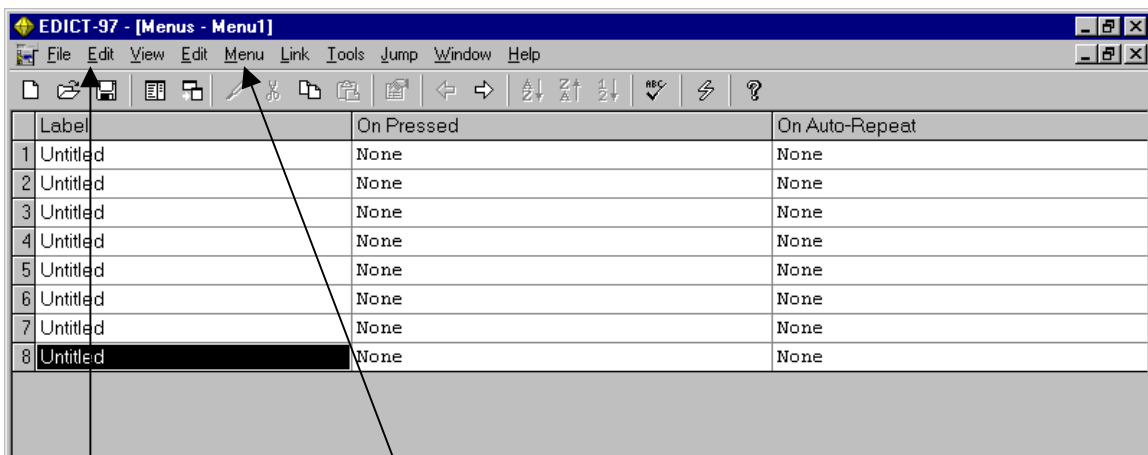
3) When there is a Comms update, the Data in Comms Block C[1] to C[5] is copied to Comms Block B[1] to B[5] respectively. (For more detail on CopyData Function, look in Function Index in Section A of this manual.)

Note on Routing. Global, routing refers to an Event having both a global and local function. If routing is Global then the local function for this event (page specific) will be followed by the Global function. System routing refers to a System key (Raise, Lower, Next, Prev, Mute, Delete, Exit, Alarms, Mute) having both System and local or global functions. If routing is System, then the local function for this event will be followed by the System function.

## Section 5 SoftKey Menus

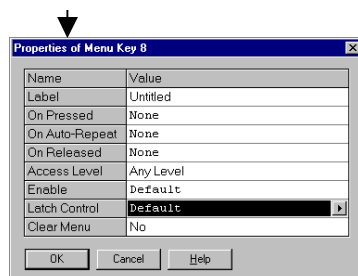
EDICT-97 has the ability to create Soft Key Menus. These menus are created in the SoftKey Menu category. The number of Soft Key Menus that can be created is limited only by the amount of memory available in the designer's database.

### SoftKey Menu Properties



Edit/properties

To add, "Go to" or rename Soft key Menus



Property	Description
Label	The text that will appear next to this Soft Key
On Pressed	The Action that results when this Soft Key is pressed. Example: gotoPage(Page2)
On Auto-Repeat	The Action that results when Soft Key is pressed and maintained. Example: count++ (the register "count" will continuously increment by 1 as long as this Soft Key press is maintained).
On Released	The Action that results when this Soft Key is released after a press. Example: <b>count:=914</b> (When this Soft Key is pressed and released the value "count" will be loaded with the value 914).
Access Level	Used with the System Security feature to control user access. Access level can be set from "Any" up through "Level 9".
Enable	An enabling expression for this Soft Key. If an expression is entered for this property, it must evaluate to a non-zero value for this Soft Key to cause the designated Action. Example: <b>Door==1</b> (The variable Door must be equal to 1 (closed) for this Soft Key to cause the designated Action.
Latch Control	An expression used to latch this Soft Key on. If an expression is entered for this property, it must evaluate to a non-zero value for this Soft Key to latch on. Example <b>Motor1ON==1</b> ( If the variable Motor1ON is equal to 1 (ON) this Soft Key will latch to the on position.
Clear Menu	Yes or No. If Yes, this Soft Key will cause this Soft Key Menu to disappear when Pressed or Released.

## Soft Key Menu Functions

**ShowMenu(Menu1)** When this function is run the SoftKey Menu 1 will slide out on a display page. Example: When this page is selected Soft Key Menu 1 will slide out.

Event	Enable	Action	Routing
1 Page selected	Default	ShowMenu(Menu1)	Default
2 None	Default	None	Default
3 None	Default	None	Default
4 None	Default	None	Default
5 None	Default	None	Default
6 None	Default	None	Default
7 None	Default	None	Default
8 None	Default	None	Default

1 Events 2 Variables

**HideMenu()** When this function is run the displayed SoftKey Menu will disappear from the display page. Example: When function key 1 is pressed the displayed SoftKey Menu will disappear.

Event	Enable	Action	Routing
1 F1 pressed	Default	HideMenu()	Default
2 None	Default	None	Default
3 None	Default	None	Default
4 None	Default	None	Default
5 None	Default	None	Default
6 None	Default	None	Default
7 None	Default	None	Default
8 None	Default	None	Default

1 Events 2 Variables

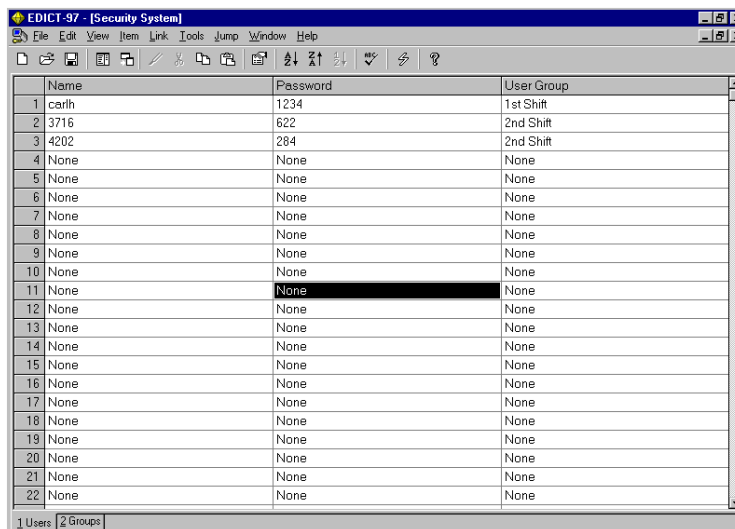
## Section 6 Security System

The system security implementation provides a fully configurable, hierarchical user specific access control solution. Access can be controlled at both the page level and at the animation level i.e. the programmer can decide which pages can be accessed (and hence viewed) and which animation items on the page the user can modify.

The hierarchical scheme has 10 access levels (0 – 9). Access is controlled by assigning an access level to the page/animation; the user must have an equivalent or higher access level to access the item.

### User Configuration

The System Security element of EDICT-97 contains the list of users; the properties are organized into user and user groups. Each user must have an entry in the **User** table, the properties in the table are unique to the user, in addition each user is assigned to a **User Group**, the group properties are common to all users of the group. **Choose the Security System Category.**



The screenshot shows a window titled "EDICT-97 - [Security System]". It contains a table with three columns: "Name", "Password", and "User Group". The table lists 22 users. The first three users have specific names and passwords, while the rest have "None" for both. The "User Group" column lists "1st Shift", "2nd Shift", and "None".

	Name	Password	User Group
1	carlh	1234	1st Shift
2	3716	622	2nd Shift
3	4202	284	2nd Shift
4	None	None	None
5	None	None	None
6	None	None	None
7	None	None	None
8	None	None	None
9	None	None	None
10	None	None	None
11	None	None	None
12	None	None	None
13	None	None	None
14	None	None	None
15	None	None	None
16	None	None	None
17	None	None	None
18	None	None	None
19	None	None	None
20	None	None	None
21	None	None	None
22	None	None	None

At the bottom of the window, there are tabs for "1 Users" and "2 Groups".

↑  
Users  
Configuration  
Window

User	Description
Name	The user name. This property is used during login to validate the user. A user name can be numeric or alphanumeric and up to eight characters in length. Up to 100 user names can be configured.
Password	The user password. This property is used during login to validate user. A user password can be up to 8 numeric characters in length.
User Group	Assigns the user to the specified user group.

The screenshot shows the EDICT-97 Security System window. It contains a table with three columns: Name, Access Level, and Idle Time. The table lists 22 groups, including shifts and numbered groups. Group 05 is highlighted. The status bar at the bottom indicates 1 User and 2 Groups.

	Name	Access Level	Idle Time
1	1st Shift	Level 1	3
2	2nd Shift	Level 2	Disabled
3	Group 05	Level 3	Disabled
4	Group 04	Level 4	Disabled
5	Group 05	Level 5	Disabled
6	Group 06	Level 6	Disabled
7	Group 07	Level 7	Disabled
8	Group 08	Level 8	Disabled
9	Group 09	Level 9	Disabled
10	Group 10	None	Disabled
11	Group 11	None	Disabled
12	Group 12	None	Disabled
13	Group 13	None	Disabled
14	Group 14	None	Disabled
15	Group 15	None	Disabled
16	Group 16	None	Disabled
17	Group 17	None	Disabled
18	Group 18	None	Disabled
19	Group 19	None	Disabled
20	Group 20	None	Disabled
21	Group 21	None	Disabled
22	Group 22	None	Disabled

Group	Description
<b>Name</b>	The name used to identify the group in the user Properties. Group names can be up to 20 Numeric/alphanumeric characters in length.
<b>Access Level</b>	Defines the access level of all users of the group.
<b>Idle Time</b>	The user timeout in seconds. This value specifies the number of seconds without a key press after which the user will be automatically logged off. If disabled the current user will remain logged on indefinitely.

## Page Access Control

The **Display Page Properties** now contain a number of **Security** properties. These Properties are used to control access to the page and the action taken in the event of access being denied.

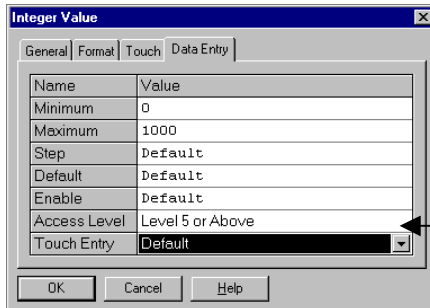
The screenshot shows the 'Display Properties' dialog box with the 'Security' tab selected. It contains a table with two columns: Name and Value. The table has three rows: Access Level (Any Level) and Fallback Page (None). There are OK, Cancel, and Help buttons at the bottom.

Name	Value
Access Level	Any Level
Fallback Page	None

<b>Access Level</b>	Defines the minimum access level required by the user to access to the page. There are 10 access levels (0-9). A group that has an access level of 5 can access pages that have Access Levels of 5 or below.
<b>Fallback Page</b>	This page is displayed if access to the page is denied. If the user does not have access to this fallback page then the fallback page of the fallback page will be displayed. This process will continue until a fallback page is found with which the user has access permission. If no default page is found in 10 attempts or no default page is specified the user will remain on the current page.

## Animation Access Control

Access to specific animations, menus etc. can be controlled where the item has an **Access Level** Property. The property specifies the minimum access level required by the user to access this item i.e. modify a value on a data entry item, press a button on a menu etc.



The 'Integer Value' dialog box has four tabs: General, Format, Touch, and Data Entry. The 'Data Entry' tab is selected. It contains a table with the following properties:

Name	Value
Minimum	0
Maximum	1000
Step	Default
Default	Default
Enable	Default
Access Level	Level 5 or Above
Touch Entry	Default

At the bottom are buttons for OK, Cancel, and Help.

Logged on user must have an Access Level of 5 or higher to access this data entry field.

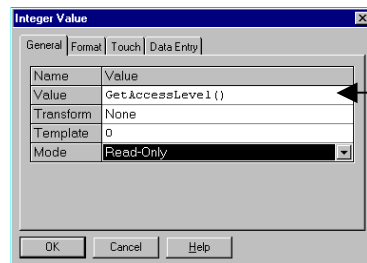
## Functions

The following functions have been added as part of the system security implementation:

**GetAccessLevel ()**  
**LogOff ()**

Returns the access level of the current User (0-9).  
Logs the current user off and sets Access level to 0.

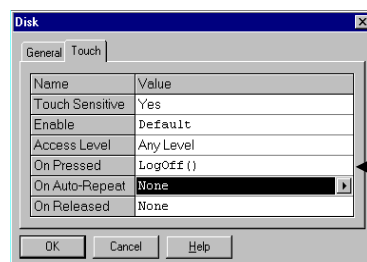
### Examples:



The 'Integer Value' dialog box, 'Data Entry' tab, showing:

Name	Value
Value	GetAccessLevel ()
Transform	None
Template	0
Mode	Read-Only

Integer Value inserted on Display page.



The 'Disk' dialog box, 'Touch' tab, showing:

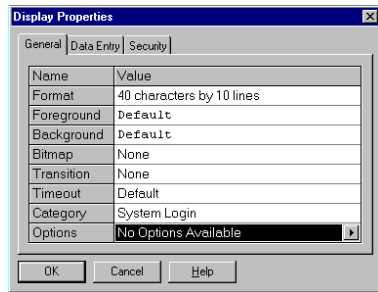
Name	Value
Touch Sensitive	Yes
Enable	Default
Access Level	Any Level
On Pressed	LogOff ()
On Auto-Repeat	None
On Released	None

Touch sensitive disk inserted on display page will Log off current user when pressed.

## Login Screen

The user can login using the system login screen, this screen has entry fields for both user name and password. The current user is logged off automatically when this page is activated i.e. no explicit logoff is required.

The system login screen is a page with Page Property **Category** set to **System Login**. It is The responsibility of the database programmer to ensure a system login screen is available and a method to access this page is provided

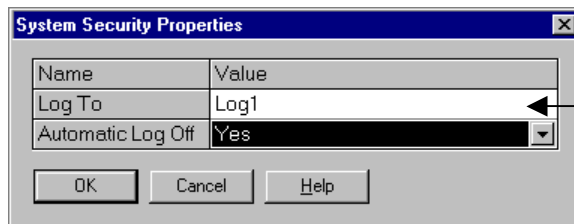


Choosing Page/Properties from the Display page toolbar accesses this window.

When users access the System Login Screen the Raise/Lower keys are used to scroll through the Users for the database. Using the Next key allows the user to enter his password. Again the Raise /Lower keys are used to enter the password (The User and Password fields on Touch units are touch sensitive and respond like data entry fields when pressed).

## System Security Properties

To access the System Security Properties choose the **Item Properties** from the **System Security** Category Page in EDICT-97.



This Log is configured in the **Events Log** category of EDICT-97.

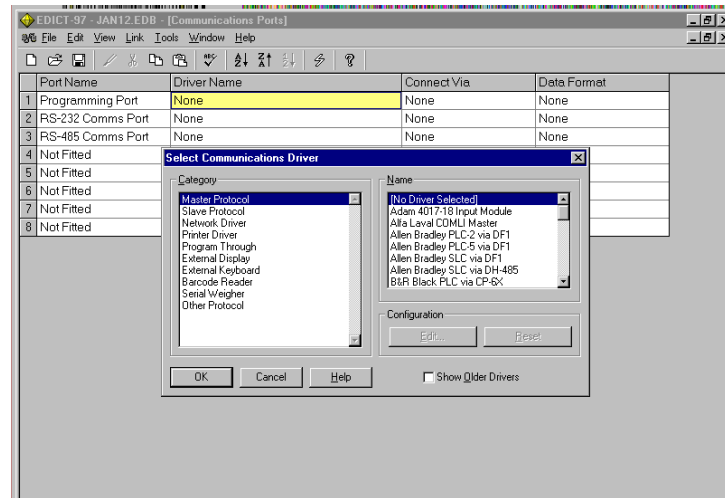
Property	Description
<b>Log To</b>	The event log. This property defines the log to which system security events are logged and enables the logging.
<b>Automatic Log Off</b>	Automatic log off on system restart. After a system restart the current user will be logged off automatically. During data base development it may be necessary to disable this action to prevent user log off during database downloads.

## Security System Events

The following table lists the events specific to system security implementation.

Event	Description
Login Successful	User attempted to login and was successful.
Login Failed	User attempted to login and failed.
User Logged Off	The current user has logged off.
User Timeout	The current user idle time has been exceeded.
Page Access Failed	The user did not have high enough access permissions to access the page.

## Section 7 Comms Ports



Depending on which model you have, there are 2 or 3 Communication Ports available on your HMI. Comm Ports are easily configured using the pull down Windows.

### Comms Port Properties

Each Comms port has the following properties...

Property	Description
Name	The name of the Comms port. This is defined by the terminal you have selected, and cannot be modified. Typically, it includes an indication of the physical standard used by the port.
Driver	The driver to be attached to the port. Expand the cell to display the driver selection dialog box. This will present you with a list of driver categories, and a list of available drivers.
Connect Via	Any link driver to be used. A link driver provides an extra level of control between the Comms driver and target device, and is used to manage modems and the like. Expand the cell to make a selection from the list of available drivers.
Data Format	The data format to be used. This setting specifies the byte format to be used on this port. This includes such things as the Baud rate and the number of data bits. Expand the cell to edit these settings. The default values are those considered to be most appropriate to the driver in question.

Please note that special considerations are required when binding a Comms driver to the Programming Port of a terminal, or it is possible to get into a situation where you are unable to program the device without first clearing its memory. When using this port, it is important that you provide a way of stopping the system and switching the port back into its usual programming role. This can be done by calling the "StopSystem" function from a suitable point in your database.

## Comms Driver Selection

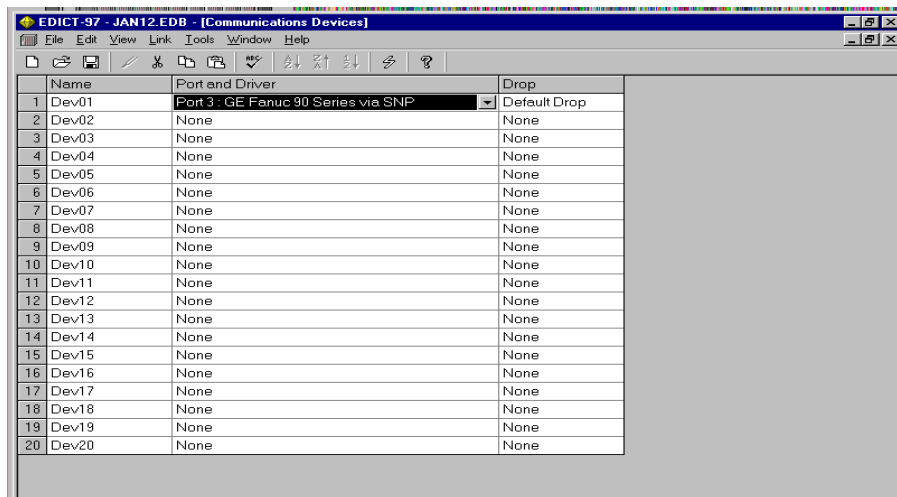
This dialog box is used to select the Comms driver to be used with the current communications port. Communications drivers are broken down into categories, with these categories being listed on the left-hand side of the dialog box. When you select a given category, the list on the right-hand side will show the available drivers.

Some Comms drivers have further configuration information, which can be edited by pressing the "Edit" button within the "Configuration" section of the dialog box. You can also restore this information to its default settings by pressing the "Reset" button in the same section. This latter button will only be enabled if changes have been made.

If you change the Comms driver on a port, which has been used, in your database, you may receive a message telling you that you need to recompile your database to handle the resulting errors. You may also find that certain Comms blocks contain addresses prefixed with "WAS", indicating that they are no longer valid for the new driver.

## Section 8 Comms Devices

After you have set up your Comms Ports, you need to set up your Comms Devices.



The example above shows how Dev01 was chosen for a GE Fanuc Series 90 PLC.

## Comms Device Properties

Each Comms device has the following properties...

Property	Description
Name	The name of the device. This name is used within the database to refer to the device. It is not "understood" by EDICT-97, and can be any string of a sensible length.
Driver	The driver used to reach this device. Each device represents a PLC or other piece of hardware to be addressed by EDICT-97, and the "Driver" entry states to which Comms port it is to be attached, and so which Comms driver is to be used.
Drop	Any drop number or network address. If the driver requires it, you may have to enter some form of address to identify the target device on the network. The driver will typically select a suitable default value if no value is entered.

## Section 9 Comms Blocks

	Device	Address	Data Type	Size	Access	Update	Enable
A	None	None	16-bit Signed	0	Read	Auto	Default
B	None	None	16-bit Unsigned	0	Read	Auto	Default
C	None	None	16-bit Signed	0	Read	Auto	Default
D	None	None	16-bit Signed	0	Read	Auto	Default
E	None	None	16-bit Signed	0	Read	Auto	Default
F	None	None	16-bit Signed	0	Read	Auto	Default
G	None	None	16-bit Signed	0	Read	Auto	Default
H	None	None	16-bit Signed	0	Read	Auto	Default
I	None	None	16-bit Signed	0	Read	Auto	Default
J	None	None	16-bit Signed	0	Read	Auto	Default
K	None	None	16-bit Signed	0	Read	Auto	Default
L	None	None	16-bit Signed	0	Read	Auto	Default
M	None	None	16-bit Signed	0	Read	Auto	Default
N	None	None	16-bit Signed	0	Read	Auto	Default
O	None	None	16-bit Signed	0	Read	Auto	Default
P	None	None	16-bit Signed	0	Read	Auto	Default
Q	None	None	16-bit Signed	0	Read	Auto	Default
R	None	None	16-bit Signed	0	Read	Auto	Default
S	None	None	16-bit Signed	0	Read	Auto	Default
T	None	None	16-bit Signed	0	Read	Auto	Default
U	None	None	16-bit Signed	0	Read	Auto	Default
V	None	None	16-bit Signed	0	Read	Auto	Default
W	None	None	16-bit Signed	0	Read	Auto	Default
X	None	None	16-bit Signed	0	Read	Auto	Default
Y	None	None	16-bit Signed	0	Read	Auto	Default

### Comms Block Properties

Each Comms block has the following properties...

Property	Description
Device	The device associated with the block. Either select the required device from the drop-down list, or leave the setting as its default value if the block is to hold internal data.
Address	The address associated with the block. For blocks linked to an external device, which organizes its data in tables, enter the start register for the block. For other devices, expand the cell and enter the address associated with each element in the block. You can also expand the cell to enter comments for each element.
Data Type	The type of data held in the block. For internal blocks, you may select any of the data types supported by EDICT-97. For blocks linked to a Comms device, EDICT-97 will restrict your choice to types supported by the Comms driver. Note that this data type will always contain at least as many bits as the underlying data type in the remote device.
Size	The number of elements in the Comms block. For internal blocks, this specifies the amount of memory to reserve. For a block associated with a Comms device, it specifies how much data is to be transferred. Note that the "Size" setting is in units of the data type specified by the previous property, and is not based upon the underlying register type.
Access	The direction of Comms transfer. For internal blocks, this setting is ignored. For external blocks, it specifies the direction in which communications is to occur. No matter what the setting of this property, you can always force updates in any direction by using the ReadBlock or WriteBlock functions.
Update	How often to update the block. For internal blocks, this setting is ignored. For external blocks, a setting of "Auto" specifies that any read operations demands are performed as soon as possible, while write operations are performed when data is changed. A setting of "Manual" indicates that no updates will occur unless manually requested, while a numeric setting indicates the minimum number of seconds to occur between updates.

Enable	An expression to enable or disable the block. If this expression is zero, the block will not be updated. This can be used when testing to disable Comms blocks until you have all the associated equipment available, or to optimize your communications by enabling or disabling blocks according to the current context.
--------	--

## Comms Block Data

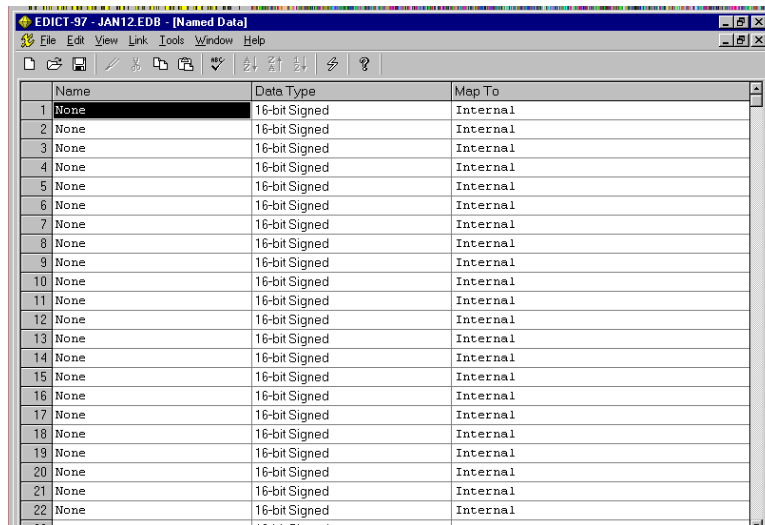
EDICT-97 allows you to configure twenty-six communication blocks, used to pass data to and from external devices such as PLCs or to store data within the terminal. Each block has a starting address and a size, and EDICT-97 will automatically transfer data to and from the block as required. The blocks are labeled from “A” to “Z”, and the elements within each block are numbered such that the first element of a block is element zero.

Comms block elements can be used within expression by following the single-letter name of the required block with an indexing expression in brackets. As an example, “A[0]” refers to the first element in block A, while “B[10]” refers to the eleventh element in Comms block B. These elements may in turn refer to registers within a remote device, depending upon how the block has been configured. If you hold your mouse pointer above an expression containing Comms block elements, EDICT-97 will display a small pop-up window listing the PLC registers being referenced.

## Indirect Addressing

As an expression is used to select the required element of a Comms block, you can use another data item to select an element depending on circumstances. This technique is known as “indirect addressing” or “indirection”, and can be used in many powerful applications. As an example, you may wish to use a single display page to show the status of several motors, with an internal variable being used to indicate which motor to display, and indirection being used to select the associated status data from a Comms block.

## Section 10 Named Data



	Name	Data Type	Map To
1	None	16-bit Signed	Internal
2	None	16-bit Signed	Internal
3	None	16-bit Signed	Internal
4	None	16-bit Signed	Internal
5	None	16-bit Signed	Internal
6	None	16-bit Signed	Internal
7	None	16-bit Signed	Internal
8	None	16-bit Signed	Internal
9	None	16-bit Signed	Internal
10	None	16-bit Signed	Internal
11	None	16-bit Signed	Internal
12	None	16-bit Signed	Internal
13	None	16-bit Signed	Internal
14	None	16-bit Signed	Internal
15	None	16-bit Signed	Internal
16	None	16-bit Signed	Internal
17	None	16-bit Signed	Internal
18	None	16-bit Signed	Internal
19	None	16-bit Signed	Internal
20	None	16-bit Signed	Internal
21	None	16-bit Signed	Internal
22	None	16-bit Signed	Internal

## Naming Data Items

EDICT-97 allows you to apply names to either existing expression or to internal memory locations within the interface terminal. These named data items are defined at either a global level, such that they are usable throughout the whole database, or at an item level, such that they can only be used within a given display page or program. Named items are divided into four categories...

Category	Description
Variable	Refers to a single writable data value, or a single register of internal memory within the terminal. Variables are considered writable, and can have values assigned to them.
Constant	Refers to an expression, which always has the same value, and contains no variables or other changeable data items. This allows EDICT-97 to optimize its evaluation.
Formula	Refers to any expression, which may or may not contain variables. Unlike variables, formulae do not have to be writable and so cannot be assigned values.
Data File	Refers to a number of data items within a Comms block, or an area of internal memory within the terminal. Each data file can contain up to 8000 registers.

Identifiers other than data files are referred to simply by their names. Data file names are normally followed by a pair of square brackets, containing an expression, which is used to determine which file element to access, with the first element being element zero. As an expression is used to select the required element, you can use a variable or any other data item to select an element depending on circumstances. This technique is known as “indirect addressing” or “indirection”, and is used in many complex applications. As an example, you may wish to use a single display page to show the status of several motors, with an internal variable being used to indicate which motor to display, and indirection being used to select the associated status data from a Comms block.

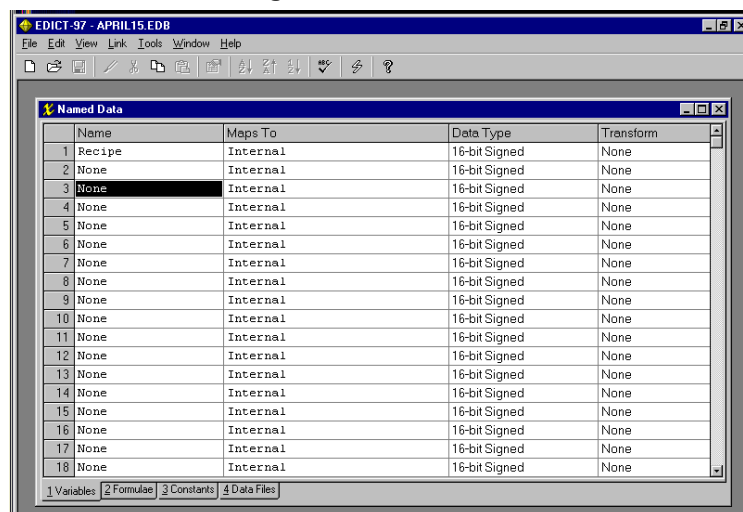
### Examples

```
TankLevel := 100           // Assignment to variable
Levels[5] := 50           // Assignment to data file element
Val[Loop] := 0            // Assignment using indirection
PI * A[0] * A[0]          // Use of constant in expression
```

### Using the Named Data Table for Recipes

The following Example shows how the Named Data Table could be used for Recipe Management. (The CommsBlocks Table could also be used for Recipe Management.)

This simple example requires that the HMI store and download 200 Recipes. Each **Recipe** has three variables: **flour**, **water** and **sugar**.



First the Named Data Table is used to declare **Recipe** as a Named Data variable.

Next the three variables: **flour**, **water**, **sugar** are declared in the **Data Files** section of the Named Data Table. Note: each of these variables is set for a size of 200 values.

Name	Maps To	Size	Data Type
1 water	Internal	200	16-bit Signed
2 flour	Internal	200	16-bit Signed
3 sugar	Internal	200	16-bit Signed
4 None	Internal	Auto	16-bit Signed
5 None	Internal	Auto	16-bit Signed
6 None	Internal	Auto	16-bit Signed
7 None	Internal	Auto	16-bit Signed
8 None	Internal	Auto	16-bit Signed
9 None	Internal	Auto	16-bit Signed
10 None	Internal	Auto	16-bit Signed
11 None	Internal	Auto	16-bit Signed
12 None	Internal	Auto	16-bit Signed
13 None	Internal	Auto	16-bit Signed
14 None	Internal	Auto	16-bit Signed
15 None	Internal	Auto	16-bit Signed
16 None	Internal	Auto	16-bit Signed
17 None	Internal	Auto	16-bit Signed
18 None	Internal	Auto	16-bit Signed

The CommsBlocks are used to download the chosen Recipe values to the PLC registers. The following window shows CommsBlock A configured for this task.

Device	Address	Data Type	Size	Access	Update	Enable
A Dev01	\$R0001	16-bit Signed	3	Both	Auto	Default
B None	None	16-bit Signed	0	Read	Auto	Default
C None	None	16-bit Signed	0	Read	Auto	Default
D None	None	16-bit Signed	0	Read	Auto	Default
E None	None	16-bit Signed	0	Read	Auto	Default
F None	None	16-bit Signed	0	Read	Auto	Default
G None	None	16-bit Signed	0	Read	Auto	Default
H None	None	16-bit Signed	0	Read	Auto	Default
I None	None	16-bit Signed	0	Read	Auto	Default
J None	None	16-bit Signed	0	Read	Auto	Default
K None	None	16-bit Signed	0	Read	Auto	Default
L None	None	16-bit Signed	0	Read	Auto	Default
M None	None	16-bit Signed	0	Read	Auto	Default
N None	None	16-bit Signed	0	Read	Auto	Default
O None	None	16-bit Signed	0	Read	Auto	Default
P None	None	16-bit Signed	0	Read	Auto	Default
Q None	None	16-bit Signed	0	Read	Auto	Default
R None	None	16-bit Signed	0	Read	Auto	Default
S None	None	16-bit Signed	0	Read	Auto	Default
T None	None	16-bit Signed	0	Read	Auto	Default

Finally the Recipe page is developed using the following windows.

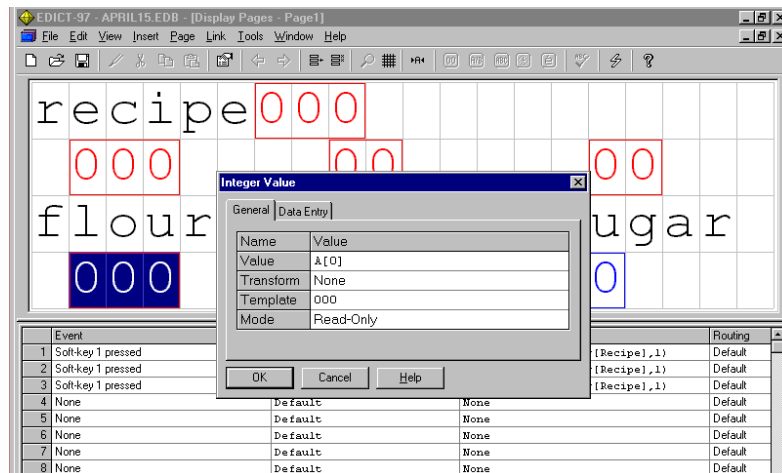
The window above shows the **Recipe Integer** .

The following window shows the variable **flour** configuration. The variables **water** and **sugar** were configured in a similar manner.

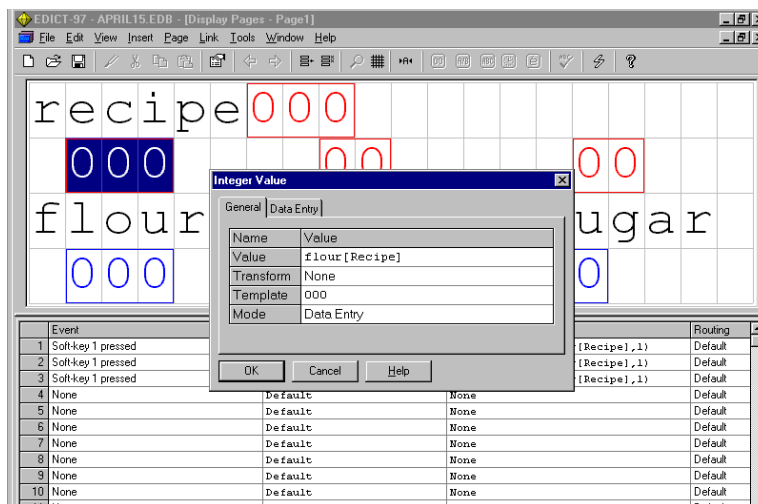
The screenshot shows a graphical user interface for configuring a recipe. It features a grid with labels for 'recipe', 'flour', and 'sugar'. Each label has associated numerical values in colored boxes (blue for 'recipe', red for 'flour', and blue for 'sugar'). A dialog box titled 'Integer Value' is open, showing configuration details for a variable named 'Value' with a 'Recipe' value, 'None' transform, '000' template, and 'Data Entry' mode. Below the grid, there is an 'Event' table and a 'Routing' table.

Event	Default	None
1 Softkey 1 pressed	Default	None
2 Softkey 1 pressed	Default	None
3 Softkey 1 pressed	Default	None
4 None	Default	None
5 None	Default	None
6 None	Default	None
7 None	Default	None
8 None	Default	None

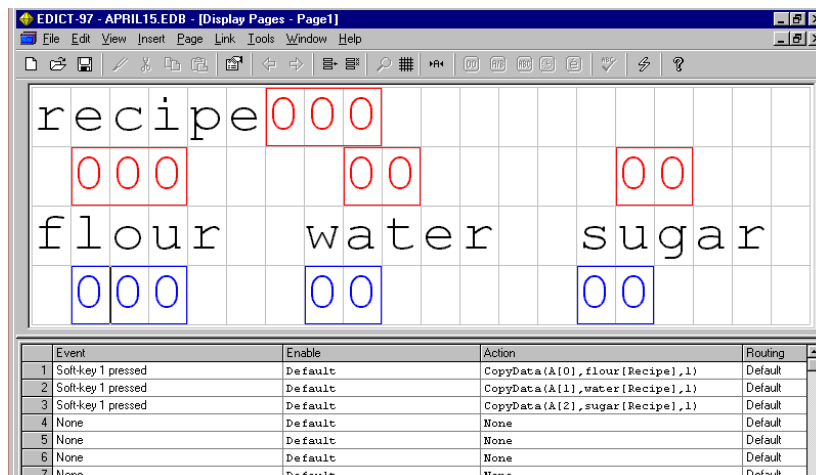
Routing	Default
[Recipe], 1	Default
[Recipe], 1	Default
[Recipe], 1	Default
[Recipe], 1	Default



The following window shows CommBlock A[0] configured to send the data to the relevant PLC register address. The CopyData function will initiate the transfer from the Recipe data file to the CommBlock A registers.



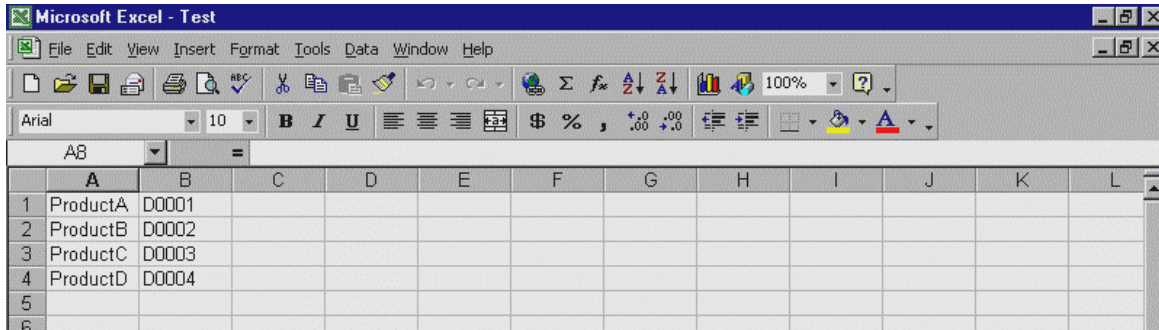
The following window shows the complete Recipe Page. Note that pressing Soft-key 1 will Copy the Recipe to the PLC addresses.



## Importing/Exporting CSV files (Comma Separated Variables)

EDICT-97 supports the Importing/Exporting of CSV files into/out of EDICT-97's Named Data table. This feature provides a seamless link between EDICT-97 and the controller's configuration software. This feature helps to reduce overall system integration time.

### Example: importing a CSV file into EDICT-97

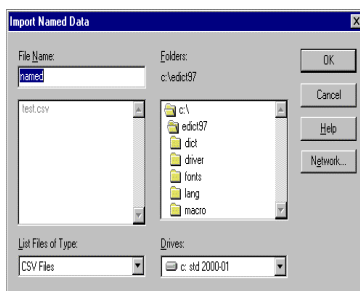
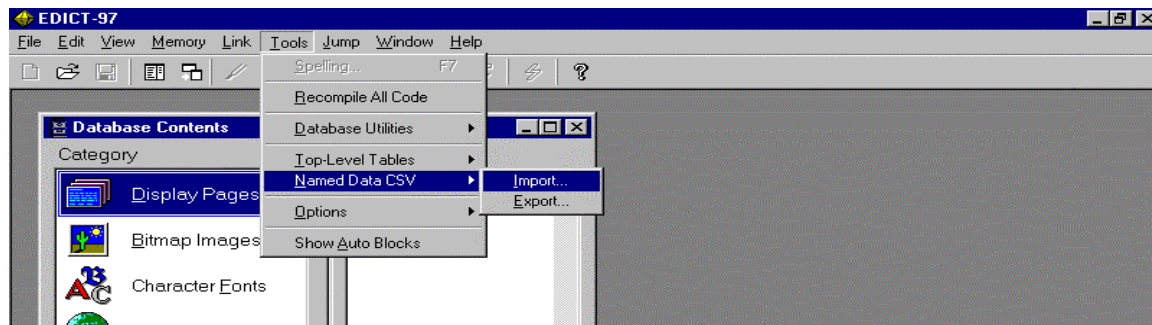


The screenshot shows a Microsoft Excel window titled "Microsoft Excel - Test". The spreadsheet has columns A through L and rows 1 through 6. The data is as follows:

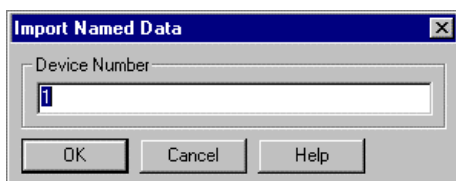
	A	B	C	D	E	F	G	H	I	J	K	L
1	ProductA	D0001										
2	ProductB	D0002										
3	ProductC	D0003										
4	ProductD	D0004										
5												
6												

Above shows a CSV file which is in the proper format to be imported into the Named Data table of an EDICT-97 database. The names of the variables are in column A and the direct references are in column B. In this example the direct references are data registers in a IDEC Mirco 3 PLC. (Note: The importing/exporting of variables into EDICT-97 works with direct references *only*.)

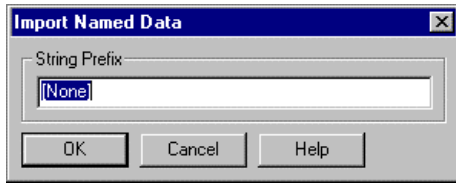
To import the CSV file into EDICT-97 do the following.



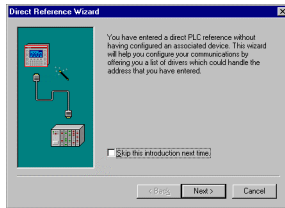
Choose the CSV file you want to import.



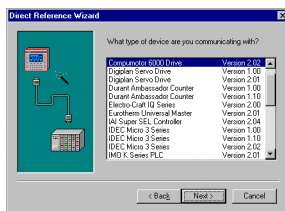
Select Device number (For multiple drops).



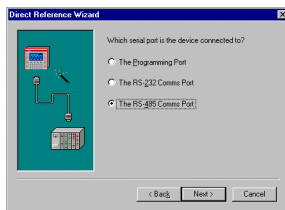
If you are using a multi-drop, you could use a prefix to differentiate between devices. For example, string prefix "D3" would alter the named data string to D3 Product A.



Direct Reference Wizard



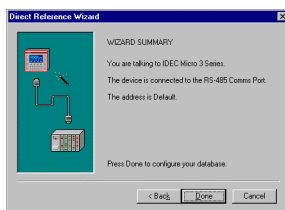
Choose Device



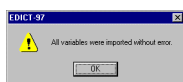
Choose Port



Choose Address of Target Device



Summary



Successful Import

The Direct PLC references are now imported into the Named Data table.

The screenshot shows the EDICT-97 software window. The title bar reads 'EDICT-97'. The menu bar includes 'File', 'Edit', 'View', 'Link', 'Tools', 'Jump', 'Window', and 'Help'. The toolbar contains various icons for file operations and editing. The main window displays a table titled 'Named Data' with the following data:

	Name	Maps To	Data Type	Transform
1	ProductA	[1,D0001]	16-bit Signed	None
2	ProductB	[1,D0002]	16-bit Signed	None
3	ProductC	[1,D0003]	16-bit Signed	None
4	ProductD	[1,D0004]	16-bit Signed	None
5	None	Internal	16-bit Signed	None
6	None	Internal	16-bit Signed	None
7	None	Internal	16-bit Signed	None
8	None	Internal	16-bit Signed	None

## **Section 11 Alarm Scanner**

### **The Alarm System**

EDICT-97 contains an alarm scanner, which continually monitors up to 500 alarm points defined in the alarm table. Each alarm point typically has a triggering expression, the state of which determines the conditions under which the alarm will be triggered. You can also indicate the mechanism to be used to accept each alarm, and whether a given alarm point will activate the terminal's internal sounder or not.

To view the list of currently active alarms, you need to create a display page with its "Category" property set to "Alarm Viewer". When this page is displayed, EDICT-97 will provide default handling for the events necessary to provide the active alarm list, to allow the operators to scroll up and down the list, and to allow alarms to be accepted. You should typically define a global event map entry to display this page when a suitable key is pressed, or when a new alarm point is activated.

If you are using the alarm system, it is also normal practice to provide a global event map entry to call the "MuteSiren" function in response to the operator pressing the Mute key on the terminal. You may choose to mute the siren in response to another event, or to perform other actions at the same time. EDICT-97's event handling system makes such customization an easy process.

If you want to operate an external siren in tandem with the terminal's internal sounder, the best technique is to place two entries in the trigger table. The first should turn on a PLC output on the rising edge on the "IsSirenOn" system variable, and the second should turn the same output off in response to a falling edge on the same variable. A similar technique can be used with "ActiveAlarms" to provide an output to drive a beacon, which remains active while any alarms are present.

### **Functions**

AcceptAlarm, AcceptAll, MuteSiren, TriggerAlarm

### **Variables**

ActiveAlarms, IsSirenOn, UnacceptedAlarms

	Alarm Name	Expression	Trigger	Accept	Priority	Print
1	Untitled	Manual	High	Manual	0	Yes
2	Untitled	Manual	High	Manual	0	Yes
3	Untitled	Manual	High	Manual	0	Yes
4	Untitled	Manual	High	Manual	0	Yes
5	Untitled	Manual	High	Manual	0	Yes
6	Untitled	Manual	High	Manual	0	Yes
7	Untitled	Manual	High	Manual	0	Yes
8	Untitled	Manual	High	Manual	0	Yes
9	Untitled	Manual	High	Manual	0	Yes
10	Untitled	Manual	High	Manual	0	Yes
11	Untitled	Manual	High	Manual	0	Yes
12	Untitled	Manual	High	Manual	0	Yes
13	Untitled	Manual	High	Manual	0	Yes
14	Untitled	Manual	High	Manual	0	Yes
15	Untitled	Manual	High	Manual	0	Yes
16	Untitled	Manual	High	Manual	0	Yes
17	Untitled	Manual	High	Manual	0	Yes
18	Untitled	Manual	High	Manual	0	Yes
19	Untitled	Manual	High	Manual	0	Yes
20	Untitled	Manual	High	Manual	0	Yes
21	Untitled	Manual	High	Manual	0	Yes
22	Untitled	Manual	High	Manual	0	Yes
23	Untitled	Manual	High	Manual	0	Yes

## Alarm Properties

Each alarm point has the following properties...

Property	Description
Alarm Title	A string used to describe the alarm, either on the in-built alarm viewer, or when logging alarms to a printer. You can enter a string of any length, but remember that it will need to fit on your terminal's display if it is to be read by the operator.
Expression	An expression which determines when the alarm is triggered, when taken together with the "Trigger" property. The expression may be left empty, if the alarm is intended to be triggered by the TriggerAlarm function.
Trigger	A multi-choice value, determining whether the alarm is level or edge triggered, and the sense in which the trigger will operate. Note that edge triggered alarms, which operate in Auto Accept mode will never persist for more than a single alarm scan.
Accept	The way in which the alarm is to be accepted. A setting of "Manual" will require the operator to accept the alarm before it can be cleared. Auto Accept alarms will clear as soon as the trigger condition is removed. This setting is rarely used with edge triggered alarms.
Priority	The priority of the alarm. This setting controls the order in which alarms are displayed by the terminal. A lower value represents a higher priority. Early versions of the runtime software may not choose to honor this value.
Print	Whether or not events associated with the alarm are printed. If this value is set to "Yes", any changes in the alarm's state will be logged to the system's default printer.
Siren	Whether or not the alarm activates the system's internal sound. A value of "Yes" will trigger the sounder whenever the alarm is triggered and remains in a non-accepted condition.

## Runtime Properties

These Panel keys perform the following specific functions when the Alarm Viewer system page is displayed on a panel (For CL,CX,VX, GL and TX series panels).

PREV key> will scroll the Alarm view back in time

NEXT key> will scroll the Alarm view forward in time

EXIT key> will exit the ALARM VIEWER system page and return to the previous page

Soft keys are configured for Accepting alarms

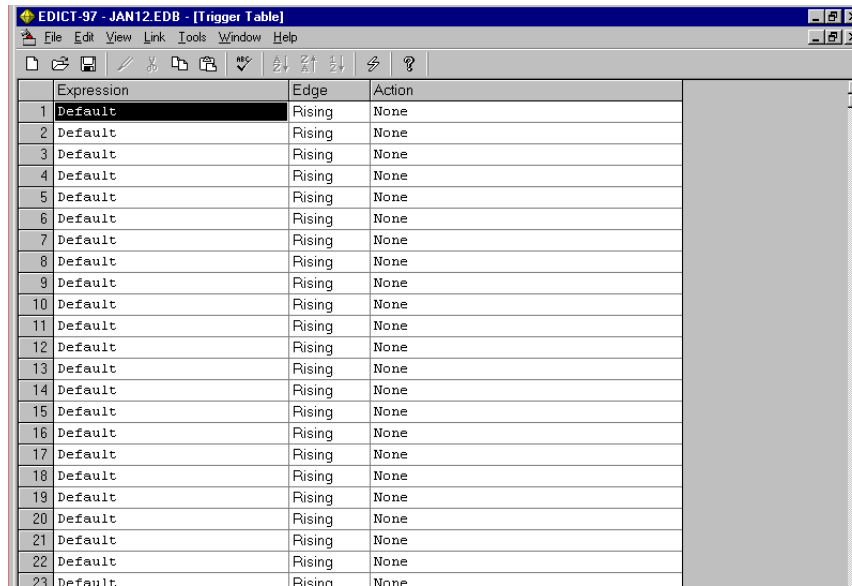
Note: For the GL300T:

- > touching the left most third of the display mimics the PREV key.
- >touching the right most third of the display mimics the NEXT key.
- >touching the central third of the display mimics the EXIT key.

## Section 12 Trigger Table

### The Trigger Table

This window is used to edit the properties of the table scanned by EDICT-97. Trigger table entries are labeled from 1 to 500, and each has a number of properties, represented by the columns of the table.



	Expression	Edge	Action
1	Default	Rising	None
2	Default	Rising	None
3	Default	Rising	None
4	Default	Rising	None
5	Default	Rising	None
6	Default	Rising	None
7	Default	Rising	None
8	Default	Rising	None
9	Default	Rising	None
10	Default	Rising	None
11	Default	Rising	None
12	Default	Rising	None
13	Default	Rising	None
14	Default	Rising	None
15	Default	Rising	None
16	Default	Rising	None
17	Default	Rising	None
18	Default	Rising	None
19	Default	Rising	None
20	Default	Rising	None
21	Default	Rising	None
22	Default	Rising	None
23	Default	Rising	None

### Trigger Properties

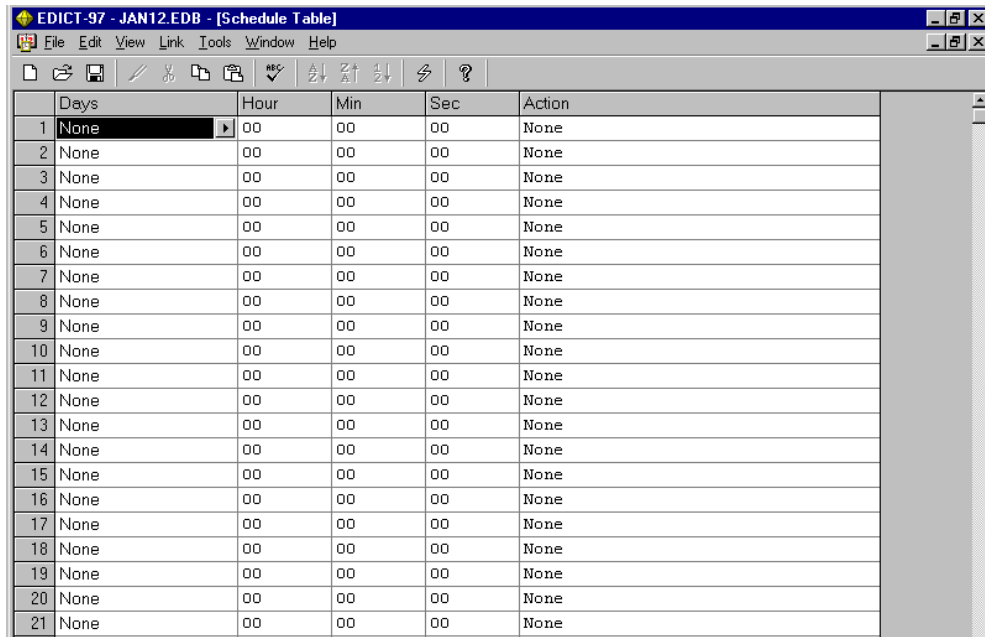
Each trigger table entry has the following properties...

Property	Description
Expression	The expression to be scanned. EDICT-97 will examine the state of this expression, and look for the transition indicated by the “Edge” property. When the appropriate edge is detected, the action specified for the trigger table entry will be executed.
Edge	The edge to be detected. If “Rising” is selected, EDICT-97 will execute the associated action when the controlling expression switches from zero to non-zero. If “Falling” is selected, the action will occur on the opposition transition.
Action	The action to be performed. This field defines what will happen when the expression changes in the required direction. You can either enter the code directly, or expand the cell and use the Action Builder to create the action for you.

## Section 13 Schedule Table

### The Schedule Table

This window is used to edit the properties of the table scanned by EDICT-97's real-time scheduler. The schedule table entries are labeled from 1 to 500, and each has a number of properties, represented by the columns of the table.



	Days	Hour	Min	Sec	Action
1	None	00	00	00	None
2	None	00	00	00	None
3	None	00	00	00	None
4	None	00	00	00	None
5	None	00	00	00	None
6	None	00	00	00	None
7	None	00	00	00	None
8	None	00	00	00	None
9	None	00	00	00	None
10	None	00	00	00	None
11	None	00	00	00	None
12	None	00	00	00	None
13	None	00	00	00	None
14	None	00	00	00	None
15	None	00	00	00	None
16	None	00	00	00	None
17	None	00	00	00	None
18	None	00	00	00	None
19	None	00	00	00	None
20	None	00	00	00	None
21	None	00	00	00	None

### Schedule Entry Properties

Each schedule table entry has the following properties...

Property	Description
Days	The days on which this entry will be activated. You can enter the initial letters of the required days separated with spaces, or one of a number of standard pattern names such as "Weekdays" and "Weekend". You can also prefix a sequence with "Not" to invert its sense. If you need to use expressions to provide more control of which days are included, expand the cell in the usual way and enter the required expressions in the resulting dialog box.
Hour	The hour at which the entry will be activated. You can enter either an expression, or an appropriate constant value. You may also enter a value of "??", to indicate that the table entry should be activated irrespective of the hour element of the current time.
Min	The minute at which the entry will be activated. You can enter either an expression, or an appropriate constant value. You may also enter a value of "??", to indicate that the table entry should be activated irrespective of the minute element of the current time.
Sec	The second at which the entry will be activated. You can enter either an expression, or an appropriate constant value. You may also enter a value of "??", to indicate that the table entry should be activated irrespective of the second element of the current time.
Action	The action to be performed. This field defines what will happen when the scheduling conditions are matched. You can either enter the code directly, or expand the cell and use the Action Builder to create the action for you.

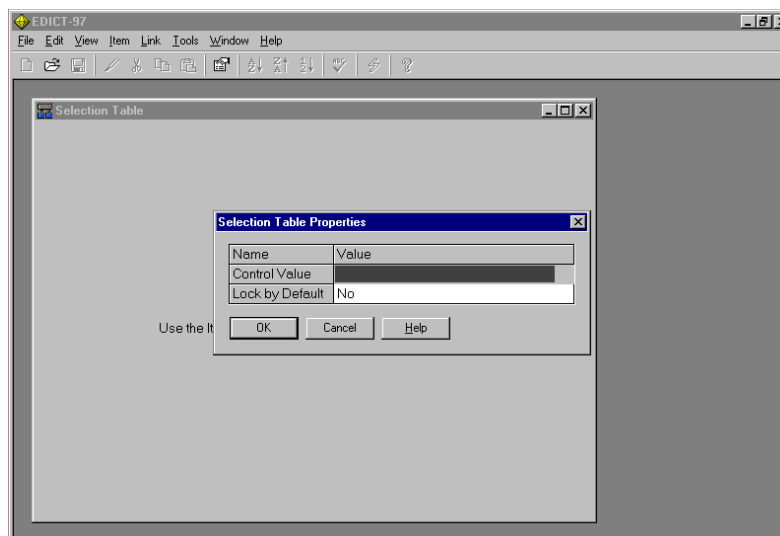
## **Section 14 Selection Table**

This window is used to edit the properties of the selection table, which allows a remote PLC or similar device to force a particular page to be displayed. The table will not be displayed until you have edited the table properties to indicate which data value should be used to select a given page. To do this, select the “Properties” command from the “Item” menu, and edit the “Control Value” property. The Selection Table contains 500 points, each of which has a number of properties, represented by the columns of the table. Please follow the link below for information about the properties of each selection table entry.

### **Selection Table Properties**

You can control the behavior of the selection table by editing its properties. These are accessed from the “Item” menu of the Selection Table window, or by right-clicking on the icon in the Database Contents window. The scanner has the following properties...

<b>Property</b>	<b>Description</b>
Control Value	The value to be used to control page selection. This should be set to an expression that EDICT-97 will compare with each row in the selection table to see if a particular page should be shown. It will normally evaluate to a register in a remote device or PLC.
Lock by Default	Whether selections will be locked by default. This controls the behavior of those pages whose “Lock” property is left at its default setting.



To access the previous Window choose **Item**, then **Properties** from the Selection Table Window. After you have entered the Control Value (for Example, you may choose A[0] which has been set in CommsBlocks to communicate to a variable memory register in a PLC, the Selection Table appears.

	Value	Page	Lock
1	As Row	Page1	Default
2	As Row	Page2	Default
3	As Row	Page3	Default
4	As Row	Page4	Default
5	3716	Page5	Default
6	As Row	None	Default
7	As Row	None	Default
8	As Row	None	Default
9	As Row	None	Default
10	As Row	None	Default
11	As Row	None	Default
12	As Row	None	Default
13	As Row	None	Default
14	As Row	None	Default
15	As Row	None	Default
16	As Row	None	Default
17	As Row	None	Default
18	As Row	None	Default
19	As Row	None	Default
20	As Row	None	Default
21	As Row	None	Default
22	As Row	None	Default
23	As Row	None	Default
24	As Row	None	Default

The example above shows 2 ways of selecting pages in the Selection table.

### **As Row**

In this example A[0] was set up to read a value in a variable register in a PLC. If the value of that register is 1, then Page1 will be selected in your HMI(a value of 2 will result in Page2 being selected, etc.)

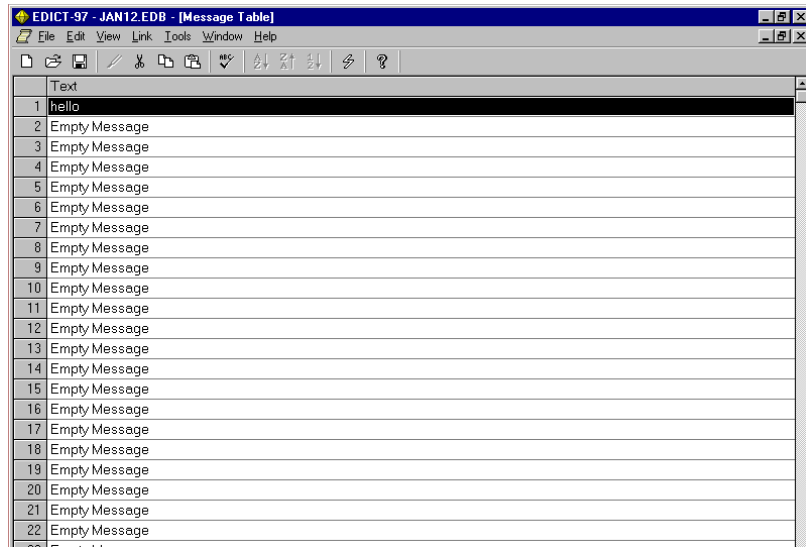
### **Specific Value**

Page 5 is selected when the value of the variable PLC register is equal to the value 3716.

## Section 15 Message Table

### The Message Table

This window is used to edit the global message table. Messages are labeled from 1 to 500.



### Message Properties

Each message has the following properties...

Property	Description
Text	The string to be shown when the message is selected for display. The string can be of any length, although the number of characters displayed will be limited by the size of the Message Text animation field.

## Section 16 Data Logger(Graphic Units)

	Channel Name	Value	Transform	Frequency	Samples	Start
1	Temperature	temp1	None	60	100	Autom
2	Untitled	Default	None	60	100	Autom
3	Untitled	Default	None	60	100	Autom
4	Untitled	Default	None	60	100	Autom
5	Untitled	Default	None	60	100	Autom
6	Untitled	Default	None	60	100	Autom
7	Untitled	Default	None	60	100	Autom
8	Untitled	Default	None	60	100	Autom
9	Untitled	Default	None	60	100	Autom
10	Untitled	Default	None	60	100	Autom
11	Untitled	Default	None	60	100	Autom
12	Untitled	Default	None	60	100	Autom
13	Untitled	Default	None	60	100	Autom
14	Untitled	Default	None	60	100	Autom
15	Untitled	Default	None	60	100	Autom
16	Untitled	Default	None	60	100	Autom
17	Untitled	Default	None	60	100	Autom
18	Untitled	Default	None	60	100	Autom
19	Untitled	Default	None	60	100	Autom
20	Untitled	Default	None	60	100	Autom
21	Untitled	Default	None	60	100	Autom

**Note:** To view all of the Channel Properties in EDICT-97( you may need to shift the Data Logger Window. Click the Start Box (Next to Samples) on the top right hand side of the Data Logger Window to view the additional properties.

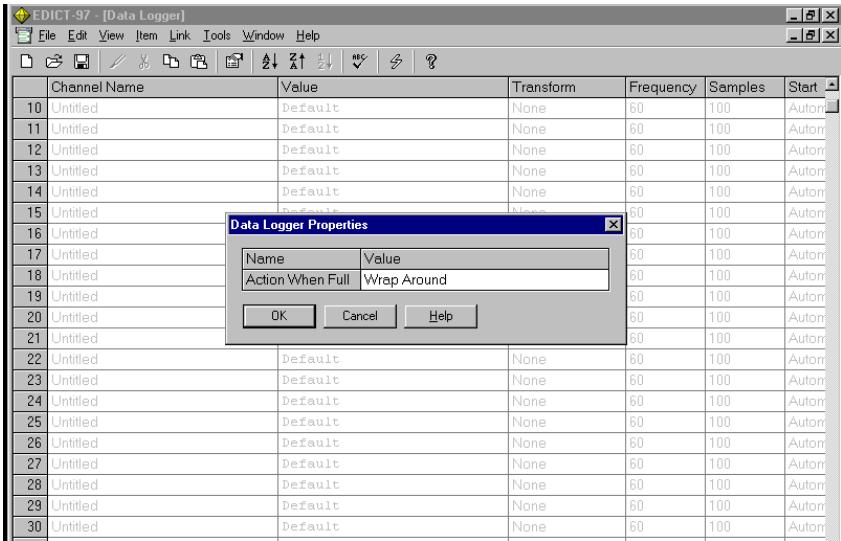
## Channel Properties

Each data logger channel has the following properties...

Property	Description
Channel Name	The name of the channel. This text will be displayed by EDICT-97 when the channel is viewed. It can also be accessed by remote devices uploading the data recorder data.
Value	The value to be recorded. This should evaluate to the expression that you wish to store in the channel. The logger records 16-bit signed values, so use the "Transform" properties to scale down any large values.
Transform	An optional transform to be performed on the data. Before the data value is stored in the logger channel, it will be transformed using the information in this property. This can be used, for example, to scale values from PLC to engineering units. Note that the "Value" property is always evaluated as a 16-bit number, and that transforms that manipulate bits will thus behave as if they have been passed a 16-bit argument, even if the underlying data is really a 32-bit value.
Frequency	How often to record a sample. This property defines the number of seconds between samples, with the minimum value being 1, and the maximum value amounting to about one and a half-hours. Heavily loaded systems may struggle to record a large number of data values at too fast a sample rate.
Samples	The size of the channel's buffer. This property indicates the size of the memory buffer allocated to this channel. When the buffer is full, the data logger will either wrap-around or stop logging, as defined by the Data Logger Properties. Each element in the buffer takes up two bytes of memory in the terminal.
Start	How the channel should be started. If this property is set to "Automatic", the channel will start automatically when the system is powered-up. If this property is set to "Manual", the TrendStart, TrendStop and TrendDefer functions can be used to control the channel manually.
Minimum	The minimum value for this channel. This value is only used when displaying the channel. It is used to set the limits on the graph used to show the channel contents. Data values are only clipped to this limit when displayed. The values in the buffer may thus be outside this limit.
Maximum	The maximum value for this channel. This value is only used when displaying the channel. It is used to set the limits on the graph used to show the channel contents. Data values are only clipped to this limit when displayed. The values in the buffer may thus be outside this limit.
Filler	The filler value for this channel. The data logger stores its data as a list of values at fixed sampling intervals, together with the time of the last reading. If the terminal is switched off for any reason, the logger must pad-out the data with a dummy value to indicate a period for which data is not available. This property defines the value to be used for this purpose, and should ideally be a value not normally found in the data being recorded.
Template	The format to be used when displaying the value. EDICT-97 uses this string when displaying the data in this channel. You can use this string to indicate the position of a fixed decimal point, or to specify units to be attached to the value. Follow the link below on Numeric Templates for more information.

To control the behavior of the Data Logger when the specified samples **Choose Item/Properties** from the top of the Data Logger Menu. The options are as follows:

Property	Description
Action When Full	What to do when a channel runs out of memory. If this property is set to "Stop Logging", the data logger will automatically stop logging to a given channel when it fills up. If it is left at "Wrap Around", the oldest data values in the buffer will be discarded and overwritten with newer values. Unless you have a good reason to change this property, it is best left at "Wrap Around".



**Note:** See Graphics Section E of this manual for additional information on creating Trend viewer pages in your database.

### Runtime Properties

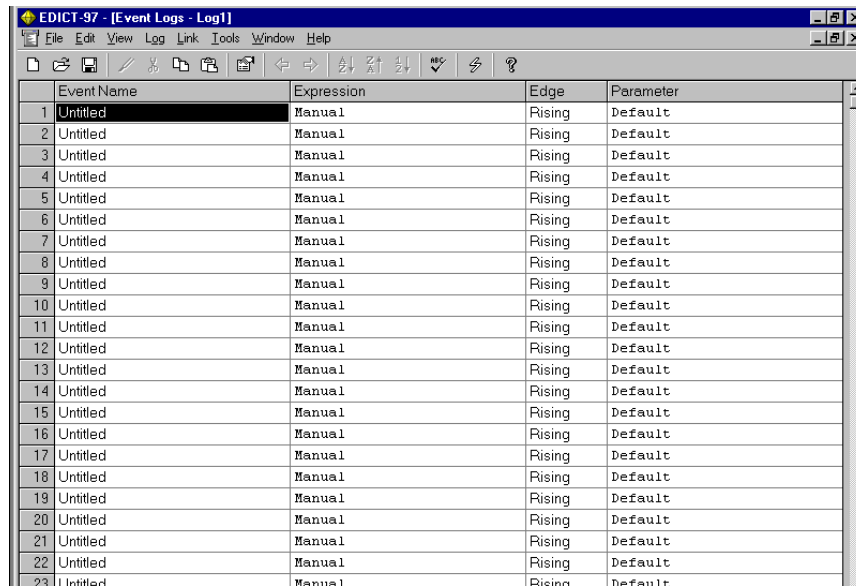
These Panel keys perform the following specific functions when the Trend Viewer system page is displayed on a panel (For VX, GL and TX series panels).

- PREV key> will scroll the Trend view back in time
- NEXT key> will scroll the Trend view forward in time
- EXIT key> will exit the TREND VIEWER system page and return to the previous page

Note: For the GL300T:

- > touching the left most third of the display mimics the PREV key.
- >touching the right most third of the display mimics the NEXT key.
- >touching the central third of the display mimics the EXIT key.

## Section 17 Event Logs



The screenshot shows the 'EDICT-97 - [Event Logs - Log1]' window. It contains a table with the following columns: Event Name, Expression, Edge, and Parameter. The table lists 23 events, all with 'Manual' as the expression and 'Rising' as the edge, and 'Default' as the parameter. The event names are 'Untitled'.

	Event Name	Expression	Edge	Parameter
1	Untitled	Manual	Rising	Default
2	Untitled	Manual	Rising	Default
3	Untitled	Manual	Rising	Default
4	Untitled	Manual	Rising	Default
5	Untitled	Manual	Rising	Default
6	Untitled	Manual	Rising	Default
7	Untitled	Manual	Rising	Default
8	Untitled	Manual	Rising	Default
9	Untitled	Manual	Rising	Default
10	Untitled	Manual	Rising	Default
11	Untitled	Manual	Rising	Default
12	Untitled	Manual	Rising	Default
13	Untitled	Manual	Rising	Default
14	Untitled	Manual	Rising	Default
15	Untitled	Manual	Rising	Default
16	Untitled	Manual	Rising	Default
17	Untitled	Manual	Rising	Default
18	Untitled	Manual	Rising	Default
19	Untitled	Manual	Rising	Default
20	Untitled	Manual	Rising	Default
21	Untitled	Manual	Rising	Default
22	Untitled	Manual	Rising	Default
23	Untitled	Manual	Rising	Default

### Event Properties

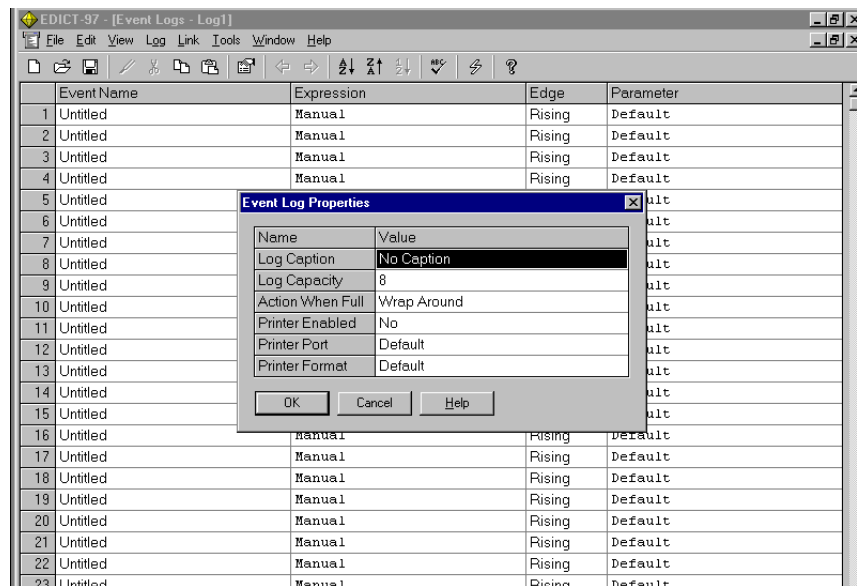
Each event has the following properties...

Property	Description
Event Name	The name of the event. This is the name that will be used when the event is to be displayed on screen or printed out. You can embed an optional numeric parameter in the event text by enclosing a valid numeric template with curly brackets. The value of the event parameter will be substituted whenever the event is displayed or printed, allowing a single event entry to record a number of different occurrences.
Expression	The expression to trigger the logging of the event. If this is left as "Manual", the event can be logged by running the LogEvent function from an event map entry or by some other means.
Edge	The edge to be detected. If "Rising" is selected, EDICT-97 will log the associated event when the controlling expression switches from zero to non-zero. If "Falling" is selected, the event will be logged on the opposition transition.
Parameter	An option parameter. EDICT-97 stores an optional 32-bit parameter with each event occurrence. This parameter can be embedded in the event name using the technique described above. This property is used to define an expression to be used as that parameter. The expression will be evaluated each time the transition defined by the "Edge" property occurs. If the event is triggered manually, the parameter is taken from the second argument to LogEvent, and this property is ignored.

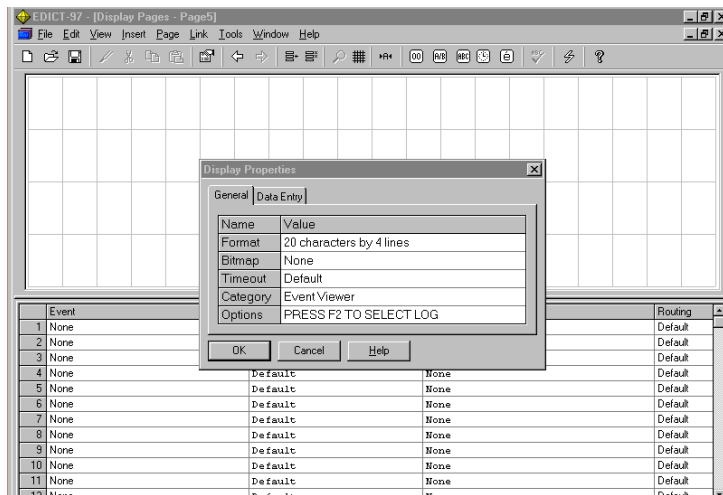
## Event Log Properties

You can control the behavior of an event log by editing its properties. These are accessed from the "Log" menu of the Event Log window, or by right clicking on the icon in the Database Contents window. The scanner has the following properties...

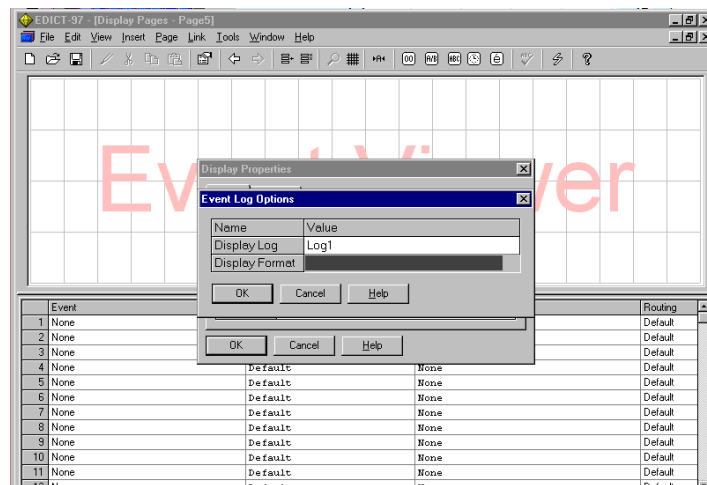
Property	Description
Log Caption	The name of the event log. This string will optionally be displayed above the event log entries when the log is displayed on screen. It can also be accessed by remote devices that wish to interrogate the event log.
Log Capacity	The number of events to store. Each event log entry takes up ten bytes of memory in the terminal. Note that the default log size is quite small to avoid a newly created Log taking up too much memory in its initial configuration. You will typically want to increase this value before using the log.
Action When Full	What to do when the log is out of memory. If this property is set to "Stop Logging", the event log will automatically stop recording events when it fills up. If it is left at "Wrap Around", the oldest events in the buffer will be discarded and overwritten with newer events. Unless you have a good reason to change this property, it is best left at "Wrap Around".
Printer Enabled	Whether or not to print the event log in real time. If this property is set to "Yes", events will be printed to the indicated printer as they are logged. Note that whatever the setting of this property, the contents of a log can be printed-out at a later point using the LogDump function.
Printer Port	The port to which events will be printed. If this option is left as "Default", events will be printed to the first port which has a printer driver bound to it. Otherwise, events will be printed to the port whose number is entered here.
Printer Format	The format to be used when printing events. If you expand this field, you will be presented with a dialog box allowing you to define the exact format to be used.



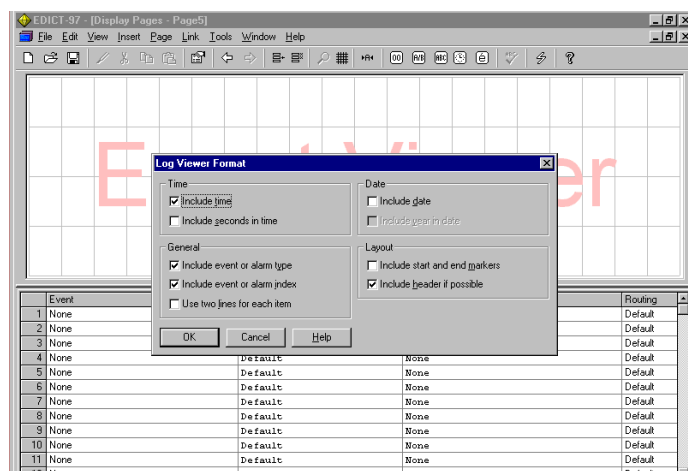
To create an **Event Viewer** Page, go to a page in the Display Pages section. Choose **Page Properties** and the following window appears.



Select **Event Viewer** for the Display Properties Category. Highlight the **Options** property and Press the **F2** key on your PC.



To set up your **Event Viewer** page display format, expand the Display Format option. The following window will appear.



## Event Formatting

You can control how events are displayed on-screen or on the printer by editing the event formatting options from this dialog box. Each check-box in the dialog box controls a given option, details of which are found below...

Option	Description
Include Time	Include the time at which the event occurred.
Include Seconds	Include the seconds portion of the event time.
Include Date	Include the date at which the event occurred.
Include Year	Include the year portion of the event time.
Include Markers	Include the start and end markers for an event log display.
Include Header	Include the event log title when viewing an event log display.
Include Type	Include a letter to indicate the type of event.
Include Index	Include the alarm or event index number.
Use Two Lines	Split each event over two lines for narrow printers or displays.

## Runtime Properties

These Panel keys perform the following specific functions when the Event Viewer system page is displayed on a panel (For CL, CX, VX, GL and TX series panels).

PREV key> will scroll the Event Log back in time

NEXT key> will scroll the Event Log forward in time

EXIT key> will exit the EVENT LOG system page and return to the previous page

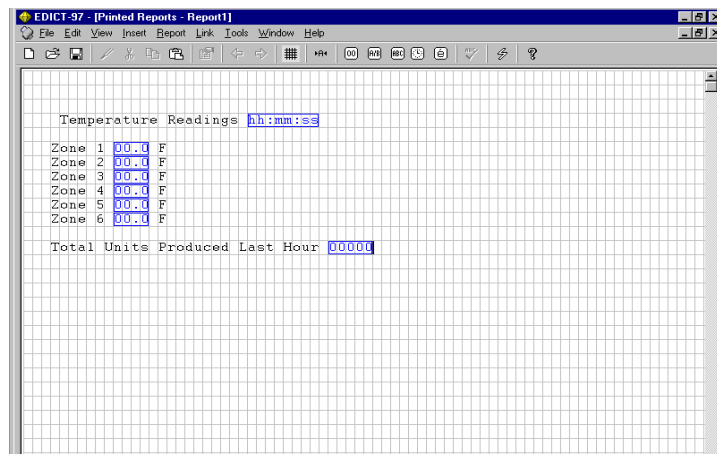
Note: For the GL300T:

- > touching the left most third of the display mimics the PREV key.
- >touching the right most third of the display mimics the NEXT key.
- >touching the central third of the display mimics the EXIT key.

## Section 18 Printed Reports

The report editor is used to define full-page reports, which can then be sent to a printer attached to one of the terminal's Comms ports. Each report may contain a combination of static text and animation items. There are various types of animation items, each capable of representing plant or internal data in a specified way. Reports are printed by calling the "PrintReport" function. This may happen in response to an entry in an event map, in response to a change in plant data, or at a given time of day on certain days of the week.

The following is a simple example.

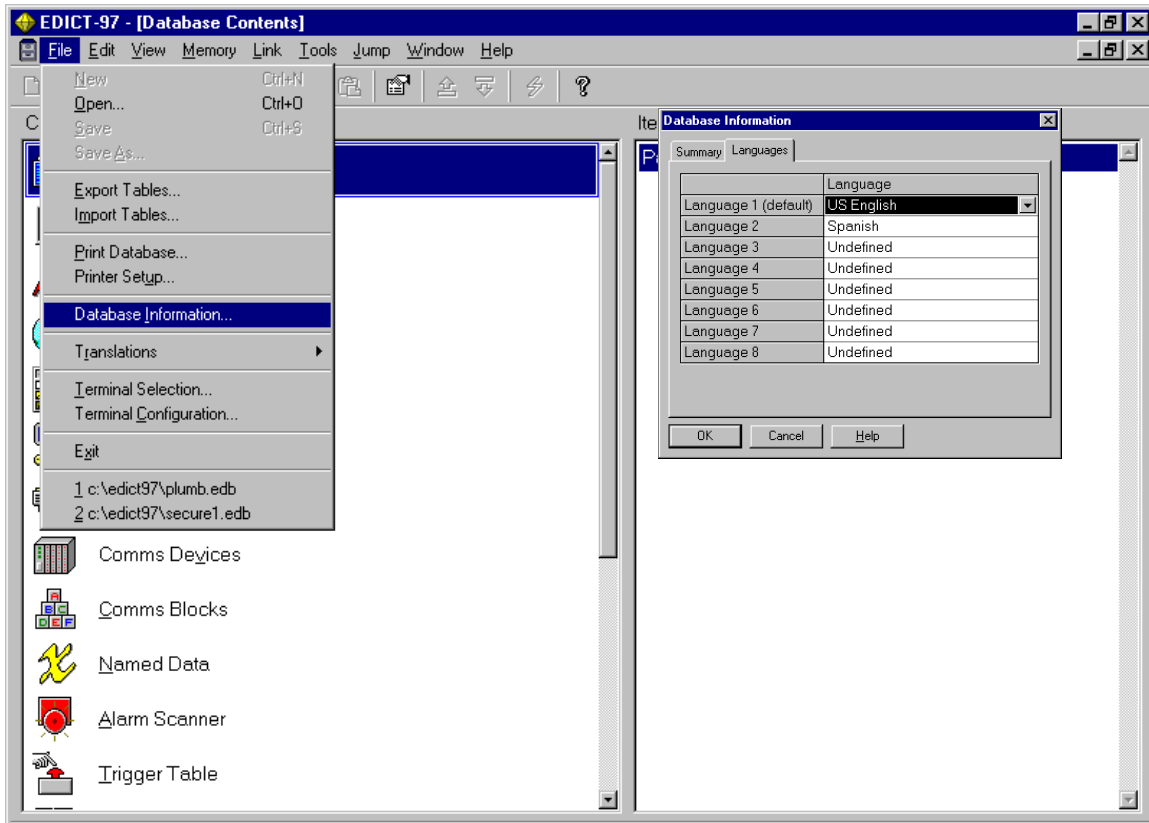


## Section 19 User Programs See Section B of this manual.

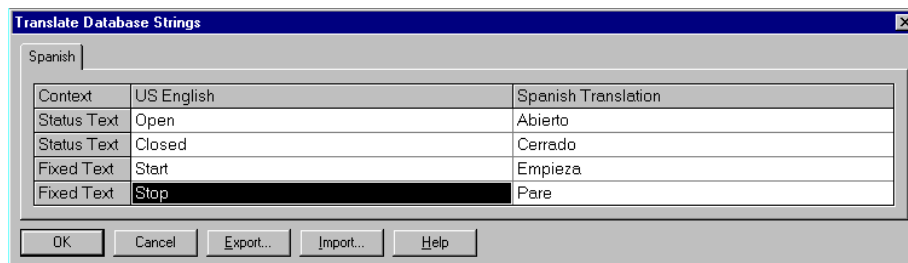
## Multiple Language Feature

The Multiple Language Feature of EDICT-97 allows the programmer to create a database where all of the text fields (See note 1) can be displayed in up to 8 different languages.

To use the Multiple Language feature, **Choose File/Database Information** from the EDICT-97 Database Contents Window.

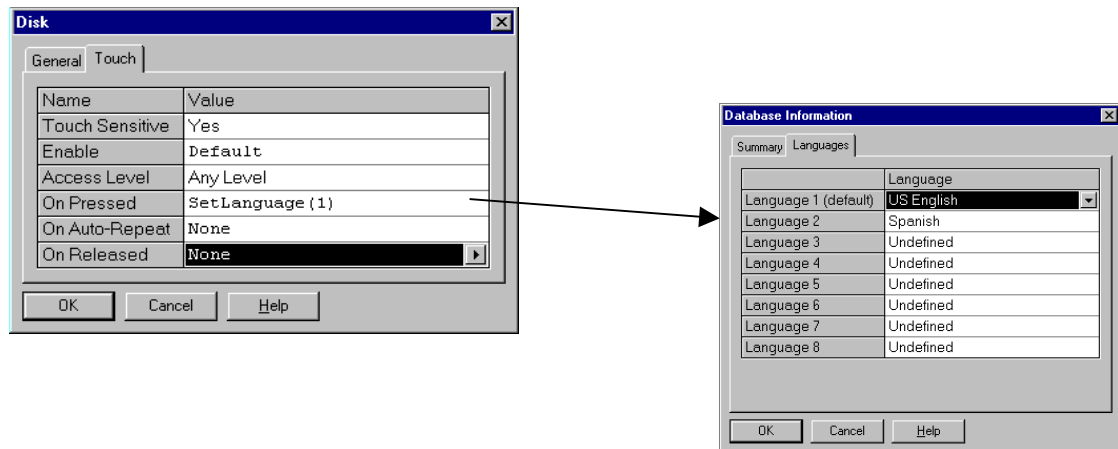


To enter the translations for your database, **Choose File/Translations/Edit Translations**.



The translations can be entered directly into the Translate Database Strings Window. Database strings can also be exported to a Text File. Using the **Export** function, the Default language file (in most cases, this would be English) is exported to a text file. The Default language is translated in the text file and imported back into the database via the **Import** function.

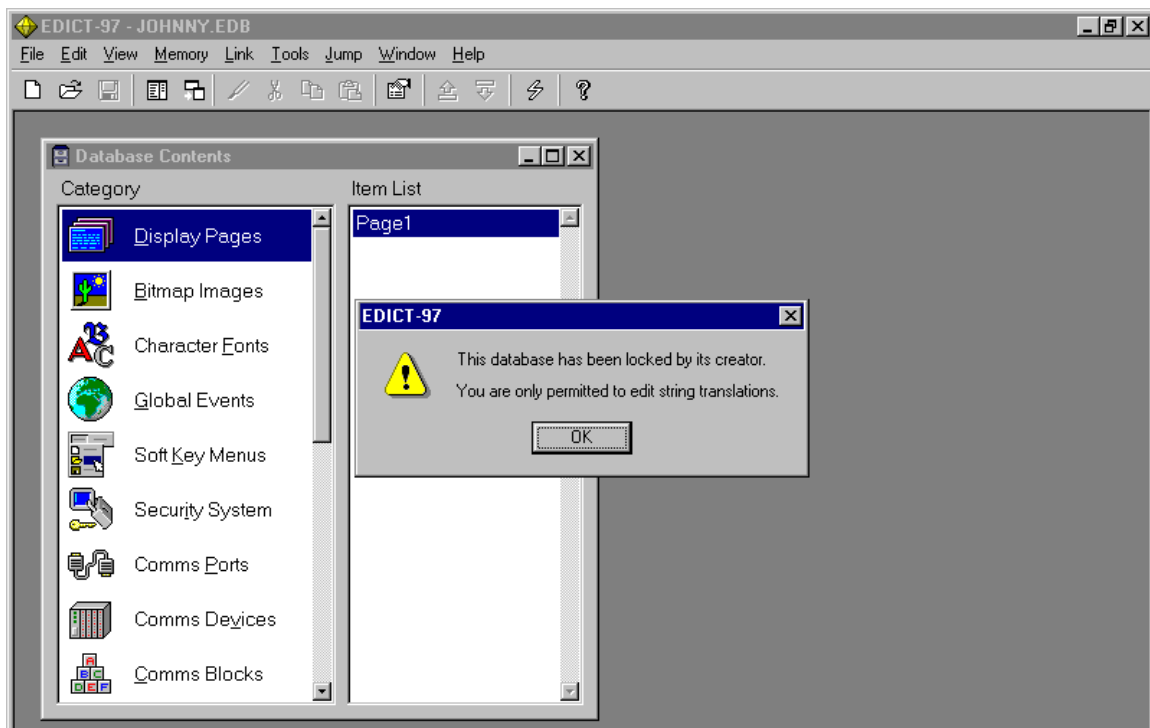
To display one of the database languages, the function **SetLanguage ()** is used.  
The following example shows a touch sensitive disk configured to select the Spanish language.



By using a **Custom Language** a database can be created where the end user has the ability to import a Custom Language. Furthermore a database can be saved as a **Locked Copy**. When the end user opens a locked copy database the only editable properties are the translation strings.

To save a database as a Locked Copy **Choose File/Translations/Save as Locked Copy**.

The following window appears when a Locked Copy database is opened.



# Section A - Functions

## Using Functions

EDICT-97 provides a number of functions, which you can call from within your programs, expressions and actions. A function is invoked by following its name with an opening round bracket, listing any arguments with comma as separators, and following the sequence with a closing round bracket. If the function takes no arguments, it must still be followed by the opening and closing brackets. EDICT-97 supports a concept called “function overloading”, where a given function may be able to take arguments of different types, and may even support optional arguments.

Functions are divided into two classes; namely, active and passive functions. An active function either changes data, or causes a change of state within EDICT-97. It can be used to form an action or to form an action statement within a program. Active functions cannot be used in expressions, where changing data is not permitted. Passive functions do not change anything within EDICT-97, but simply return a value based upon their arguments. For example, Min is a function, which returns the lower of its two arguments. Passive functions are allowed within expressions, but may not, on their own, form actions.

## Examples

```
GotoPage(Page1)      // Function invocation in an action
StopSystem();        // Function invocation as program statement
Min(A[1], A[2])      // Function invocation in an expression
```

## Index of Functions

<b>Advanced Math Functions</b>	Page A3
<b>Abs(Value)</b>	Page A6
<b>AcceptAlarm(Index)</b>	Page A6
<b>AcceptAll( )</b>	Page A7
<b>Beep(Frequency, Period)</b>	Page A7
<b>CallFloat(Program)</b>	Page A8
<b>CallInt(Program)</b>	Page A8
<b>CallLong(Program)</b>	Page A9
<b>CallString(Program)</b>	Page A9
<b>ClearRx(Port)</b>	Page A10
<b>ClearTx(Port)</b>	Page A10
<b>CopyData(Dest,Source,Count)</b>	Page A11
<b>Date(Year, Month, Date)</b>	Page A11
<b>Dispatch(Program)</b>	Page A12
<b>Fill(Dest,Count)</b>	Page A12
<b>FlipEntrySign()</b>	Page A13
<b>Format(Template,Value)</b>	Page A13
<b>GetMessage(Index)</b>	Page A14
<b>GetNow()</b>	Page A14
<b>GetTimer()</b>	Page A15
<b>GotoField(Set)</b>	Page A15
<b>GotoPage(Page)</b>	Page A16
<b>GotoPrevious()</b>	Page A16
<b>HexVal(Text)</b>	Page A17
<b>HideMenu()</b>	Page A17
<b>HoldTx(Port,State)</b>	Page A17
<b>IsMenuActive()</b>	Page A18
<b>IsOnLine(Port)</b>	Page A18
<b>Left(Text,Count)</b>	Page A19
<b>Len(Text)</b>	Page A19
<b>LogClear(Log)</b>	Page A20
<b>LogDump(Log, Port, Count)</b>	Page A20
<b>LogEvent(Log, Code, Param)</b>	Page A20
<b>LogOff()</b>	Page A20
<b>MakeChar(Value)</b>	Page A21
<b>Max(Value1,Value2)</b>	Page A21

<b>Mean(Data,Count)</b>	Page A22
<b>Mid(Text,Offset,Count)</b>	Page A22
<b>Min(Value1,Value2)</b>	Page A23
<b>ModemAnswer(Port)</b>	Page A23
<b>ModemDial(Port, Number)</b>	Page A23
<b>ModemHangUp(Port)</b>	Page A23
<b>ModemRinging(Port)</b>	Page A23
<b>MuteSiren()</b>	Page A24
<b>PopDev(Data,Count)</b>	Page A24
<b>PostEvent(Param)</b>	Page A25
<b>PrintFormFeed(Port)</b>	Page A25
<b>PrintMessage(Port,Message)</b>	Page A26
<b>PrintNewLine(Port)</b>	Page A26
<b>PrintReport(Port,Report)</b>	Page A27
<b>PrintString(Port,String)</b>	Page A27
<b>PrintTimeStamp(Port)</b>	Page A28
<b>Random(Range)</b>	Page A28
<b>ReadBlock(Block)</b>	Page A28
<b>RelayClose()</b>	Page A29
<b>RelayOpen()</b>	Page A29
<b>Right(Text,Count)</b>	Page A29
<b>Run(Program)</b>	Page A30
<b>SerialPrint(Port,Text)</b>	Page A30
<b>SerialRead(Port)</b>	Page A31
<b>SerialWrite(Port)</b>	Page A31
<b>SetBreak(Port,State)</b>	Page A32
<b>SetCommsTask(Port,Program)</b>	Page A33
<b>SetLanguage(Language)</b>	Page A33
<b>SetRTS(Port,State)</b>	Page A34
<b>SetTimer(Value)</b>	Page A34
<b>Sgn(Value)</b>	Page A35
<b>ShowMenu(Menu)</b>	Page A35
<b>SirenOn()</b>	Page A36
<b>Sleep(Period)</b>	Page A36
<b>Sqrt(Value)</b>	Page A37
<b>StdDev(Data,Count)</b>	Page A37
<b>StopSystem()</b>	Page A38
<b>Time( Hour, Minute,Second)</b>	Page A38
<b>ToggleMenu(Menu)</b>	Page A39
<b>TrendClear(Channel)</b>	Page A39
<b>TrendDefer(Channel, Time)</b>	Page A39
<b>TrendStart(Channel)</b>	Page A39
<b>TrendStop(Channel)</b>	Page A39
<b>TriggerAlarm(Index)</b>	Page A40
<b>UseAutoEnables(Port)</b>	Page A40
<b>UseHalfDuplex(Port)</b>	Page A41
<b>Val(Text)</b>	Page A41
<b>WriteBlock(Block)</b>	Page A42

Note: Experienced 'C' programmers may find it useful to peruse the file C:\Edict97\sysfunc.inf for information on various things that can be done in programming. This file comprises declarations for all the functions available in Edict97.

## Advanced Math Functions

### Sin(*Value*)

Argument	Type	Description
Value	Floating point	Returns the trigonometric sine of the angle <i>Value</i> . The <i>Value</i> of the angle must be in radians.

### Cos(*Value*)

Argument	Type	Description
Value	Floating point	Returns the trigonometric cosine of the angle <i>Value</i> . The <i>Value</i> of the angle must be in radians.

### Tan(*Value*)

Argument	Type	Description
Value	Floating point	Returns the trigonometric tangent of the angle <i>Value</i> . The <i>Value</i> of the angle must be in radians.

### Arcsin(*Value*)

Argument	Type	Description
Value	Floating point	Returns the angle in radians for the floating point <i>Value</i> equivalent to the sine of that angle.

### ArcCos(*Value*)

Argument	Type	Description
Value	Floating point	Returns the angle in radians for the floating point <i>Value</i> equivalent to the cosine of that angle.

### Arctan(*Value*)

Argument	Type	Description
Value	Floating point	Returns the angle in radians for the floating point <i>Value</i> equivalent to the tangent of that angle.

## Return type

These functions return floating point values.

## Examples

For the angle  $\theta$  using Real Numbers inserted on a display page .

If  $\theta$  is equal to 0.79 radians

$\text{Sin}(\theta)$

Returns 0.71

$\text{Cos}(\theta)$

Returns 0.71

$\text{Tan}(\theta)$

Returns 1.00

$\text{ArcSin}(0.71)$

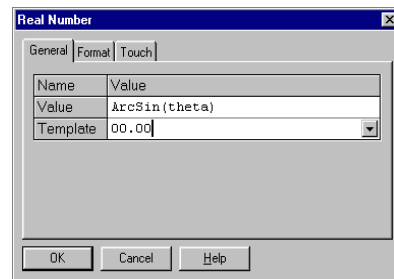
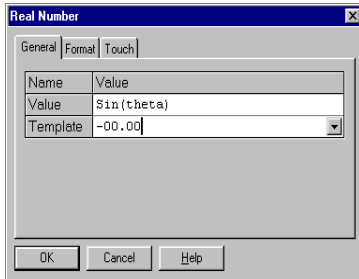
Returns .79 radians

$\text{ArcCos}(0.71)$

Returns .79 radians

$\text{ArcTan}(1.00)$

Returns .79 radians



## Log10(Value)

Argument	Type	Description
Value	Floating point	Returns the log of the floating point number <i>Value</i> .

## Examples

For the floating point number *Values* 5.0, 8.0

$\text{Log10}(5.0)$  returns the floating point number 0.699

$\text{Log10}(8.0)$  returns the floating point number 0.903

## Exp10(Value)

Argument	Type	Description
Value	Floating point	Returns 10 raised to the floating point number <i>Value</i> .

## Examples

For the floating point number *Values* 0.699, 0.903

$\text{Exp10}(0.699)$  returns the floating point number 5.0

$\text{Exp10}(0.903)$  returns the floating point number 8.0

## Log(*Value*)

Argument	Type	Description
Value	Floating point	Returns the natural log of the floating point number <i>Value</i> .

### Examples

For the floating point number *Values* 5.0, 8.0

Log(5.0) returns the floating point number 1.6094

Log(8.0) returns the floating point number 2.0794

## Exp(*Value*)

Argument	Type	Description
Value	Floating point	Returns $e$ (2.7183) raised to the floating point number <i>Value</i> .

### Examples

For the floating point number *Values* 1.6094, 2.0794

Exp(1.6094) returns the floating point number 5.0

Exp(2.0794) returns the floating point number 8.0

## Power(*Value*, *Power*)

Argument	Type	Description
Value, Power	Both Floating point	Returns the floating point <i>Value</i> raised to specified <i>Power</i> .

### Examples

For the floating point numbers *Value*=10.0 *Power*=3.0

Power(10.0, 3.0) returns the floating point number 1000.0

## MakeFloat(*ulong value*)

Argument	Type	Description
Value	32 bit unsigned	Converts the Value to a real number based on the IEEE-754 standard.

This function is useful for taking a 32 bit value representative of an IEEE-754 floating point number, and converting it into a real number in the terminal.

## Examples

MakeFloat( 0xC1480000) returns the value –12.5.

## FromFloat(float)

Argument	Type	Description
Value	Float	Converts the Value to a 32 bit number conforming to the IEEE-754 standard.

## Overview

This function is useful for taking a real number in the terminal, and converting it to the 32 bit IEEE-754 representation that some devices require for floating point operations.

## Examples

FromFloat(-12.5) will return the value 0xC1480000.

## Abs(Value)

Argument	Type	Description
Value	Numeric	The value to be processed.

## Overview

This function returns the absolute value as its argument.

## Description

If the argument to this function is negative, the function returns the positive equivalent value. If the value is positive or zero, the function simply returns that value. Calling the function with an unsigned argument will return produce the value that was passed.

## Return Type

This function returns a value of the same type as its argument.

## Examples

Event	Enable	Action
Comms Update	Default	A[0] := Abs(A[1])

After every comms scan, this will set A[0] equal to the absolute value of A[1].

## AcceptAlarm(Index)

Argument	Type	Description
Index	16-bit Unsigned	The alarm number to be accepted.

## Overview

This function accepts the alarm indicated by the *Index* argument.

## Description

The *Index* parameter should be a number between 1 and 500, and should refer to an alarm within the database's Alarm Table. If the alarm has already been accepted, is an auto-accept alarm or is not active, this function will not perform any action. If the alarm is accepted as a result of this function, an "Alarm Accepted" event will be posted.

### Return Type

This function does not return a value.

### Examples

Event	Enable	Action
Soft-Key 1 Pressed	Default	AcceptAlarm(1)

This will accept Alarm 1 without having to go to the Alarm Viewer page.

### AcceptAll()

Argument	Type	Description
None		

### Overview

This function accepts all non-accepted active alarms.

### Description

This function will accept any alarms, which are capable of being accepted. For each alarm that is accepted, an "Alarm Accepted" event will be posted. If there are many alarms to be accepted, the system's event queue may overflow, and not all events will be processed.

### Return Type

This function does not return a value.

### Examples

Event	Enable	Action
Soft-Key 1 Pressed	Default	AcceptAll()

This will accept all alarms without having to go to the Alarm Viewer Page.

### Beep(*Frequency, Period*)

Argument	Type	Description
Frequency	16-bit Unsigned	The required frequency in semitones.
Period	16-bit Unsigned	The required length in milliseconds.

### Overview

This function causes the terminal's beeper to make a sound.

### Description

This function turns on the terminal's beeper at the specified frequency for the specified number of milliseconds. Passing a value of zero for the *Period* argument will turn off the beeper. Beep requests are not queued, so calling the function will immediately override any previous calls. For those of you with a musical bent, the *Frequency* argument is calibrated in semitones, with a value of 24 being tuned to middle A. You can combine this function with the Sleep function to create programs which play tunes, should you find that you have nothing better to do with your time. On a more serious "note", the Beep function can be a useful debugging aid, as it provides an asynchronous method of signaling the handling of an event, or the execution of a program step. This technique was actually used when developing the low-level EDICT-97 system software.

### Return Type

This function does not return a value.

## Examples

Event	Enable	Action
Page Selected	Speed >= 100	Beep(48, 100)

This will “beep” if the variable “Speed” is detected as greater than 99 when the page is first displayed.

## CallFloat(*Program*)

Argument	Type	Description
Program	Program Name	The program to be executed.

### Overview

This function calls the indicated program, expecting a floating point value to be returned.

### Description

This function executes the indicated program, and returns a value equal to that passed to the return statement, or zero if no argument is passed to that statement, or if execution simply “falls off” the end of the program. If the data type of the returned value does not match that expected, EDICT-97 will attempt a conversion. If a conversion cannot be performed, a zero value will be returned.

Note that this function can be called in all contexts, even those mandating that no values be changed or actions performed. If the program you execute changes values that effect the context from which the function is called, EDICT-97’s behavior in terms of processing these changes is undefined. Likewise, calling function which change the current page will produce undefined behavior.

### Return Type

This function returns a floating point value.

## Examples

### USER PROGRAM “FIND AREA”

...

Area := Length \* Width;

return Area;

Event	Enable	Action
F1 Pressed	Default	A[0] := CallFloat(FindArea)

This will set A[0] to the floating point value calculated in the User Program.

## CallInt(*Program*)

Argument	Type	Description
Program	Program Name	The program to be executed.

### Overview

This function calls the indicated program, expecting a 16-bit value to be returned.

### Description

This function executes the indicated program, and returns a value equal to that passed to the “return” statement, or zero if no argument is passed to that statement, or if execution simply “falls off” the end of the program. If the data type of the returned value does not match that expected, EDICT-97 will attempt a conversion. If a conversion cannot be performed, a zero value will be returned.

Note that this function can be called in all contexts, even those that mandating that no values be changed or actions performed. If the program you execute changes values that affect the context from which the function is called, EDICT-97’s behavior in terms of processing these changes is undefined. Likewise, calling functions that change the current page will produce undefined behavior.

### Return Type

This function returns a 16-bit signed value.

### Examples

#### USER PROGRAM “FindSpeed”

...

Speed := Distance / Time ;

return Speed;

Event	Enable	Action
F1 Pressed	Default	A[0] := CallInt(FindSpeed)

When F1 is pressed, A[0] will be loaded with the Integer value of Speed.

### CallLong(*Program*)

Argument	Type	Description
Program	Program Name	The program to be executed.

### Overview

This function calls the indicated program, expecting a 32-bit value to be returned.

### Description

This function executes the indicated program, and returns a value equal to that passed to the “return” statement, or zero if no argument is passed to that statement, or if execution simply “falls off” the end of the program. If the data type of the returned value does not match that expected, EDICT-97 will attempt a conversion. If a conversion cannot be performed, a zero value will be returned.

Note that this function can be called in all contexts, even those mandating that no values be changed or actions performed. If the program you execute changes values that effect the context from which the function is called, EDICT-97’s behavior in terms of processing these changes is undefined. Likewise, calling functions that change the current page will produce undefined behavior.

### Return Type

This function returns a 32-bit signed value.

### Examples

#### USER PROGRAM “FindSize”

Size := Length \* Width \* Height;

return Size;

Event	Enable	Action
F1 Pressed	Default	A[0] := CallLong(FindSize)

This will set A[0] to a 32 bit number equal to the calculated value “Size”.

### CallString(*Program*)

Argument	Type	Description
Program	Program Name	The program to be executed.

### Overview

This function calls the indicated program, expecting a string to be returned.

### Description

This function executes the indicated program, and returns a value equal to that passed to the “return” statement, or an empty string if no argument is passed to that statement, or if execution simply “falls off” the end of the program. If the data type of the returned value does not match that expected, EDICT-97 will attempt a conversion. If a conversion cannot be performed, an empty string will be returned.

Note that this function can be called in all contexts, even those mandating that no values be changed or actions performed. If the program you execute changes values that effect the context from which the function is called, EDICT-97’s behavior in terms of processing these changes is undefined. Likewise, calling functions that change the current page will produce undefined behavior.

### Return Type

This function returns a string.

### Examples

#### USER PROGRAM “FormatValue”

```
Value := “Hello World!”;  
return Value;
```

Event	Enable	Action
F1 Pressed	Default	Text := CallString(FormatValue)

This will set the string variable “Text” to be Hello World!.

### ClearRx(*Port*)

Argument	Type	Description
Port	16-bit Unsigned	The required serial port.

### Overview

This function clears the receive buffer of the indicated serial port.

### Description

This function should only be called for a port to which the Roll-Your-Own Protocol driver has been bound. It will discard any characters received on the port and currently held in memory. It is often called before starting a Comms interaction to avoid any “leftovers” from a previous attempt; confusing the receive routine.

### Return Type

This function does not return a value.

### Examples

Event	Enable	Action
System Initialized	Default	ClearRx(1)

This will clear the receiver on port 1 (programming port) when the system is first started..

### ClearTx(*Port*)

Argument	Type	Description
Port	16-bit Unsigned	The required serial port.

### Overview

This function clears the transmit buffer of the indicated serial port.

### Description

This function should only be called for a port to which the Roll-Your-Own Protocol driver has been bound. It will discard any characters awaiting transmission on the port, apart from those characters that have already been passed-on to the port hardware. If “auto-enables” has been selected and RTS has been turned off, the control line will be allowed to revert to its unasserted state once any remaining characters have been sent.

### Return Type

This function does not return a value.

### Examples

Event	Enable	Action
All Input Complete	Default	ClearTx(1)

This will clear the transmit buffer for port 1, after data entry is complete, in preparation for transmitting.

### CopyData(*Dest, Source, Count*)

Argument	Type	Description
Destination	Data Reference	First item of destination data.
Source	Data Reference	First item of source data.
Count	16-bit Unsigned	Number of data items to copy.

### Overview

This function copies *Count* data items from *Source* to *Destination*.

### Description

This function copies data as described above, where *Source* and *Destination* are both references to array elements of the same basic type. This implies that both must refer to 16-bit integers, 32-bit integers, floating-point values or strings. The signed or unsigned attributes of the data do not matter, nor does the length of the strings. If you try to read beyond the end of an array, a zero or empty value will be copied.

### Return Type

This function does not return a value.

### Examples

Event	Enable	Action
Comms Update	Default	CopyData(A[0], B[0], 10)

This will copy 10 items from Comms Block B[0] through B[9] to Comms Block A[0] through A[9] at the end of a communications scan.

### Date(*Year, Month, Date*)

Argument	Type	Description
Year	16 bit unsigned	Number of the year.
Month	16 bit unsigned	Number of the month.
Date	16 bit unsigned	Number of the date.

### Overview

This function generates a 32 bit number that represents the number of seconds from Jan. 1 1997.

### Description

Calling Date() after assigning values to Year, Month, and Date, will give a 32 bit unsigned number that represents the number of seconds from Jan 1 1997 to the beginning of the date given. See also Time().

### Return Type

This function returns a 32 bit unsigned number.

### Examples

Event	Enable	Action
F1 Pressed	Default	TrendDefer( 1, Date( 2010, 1, 1 ) )
The above line will start trending on channel 1 on Jan 1 2010.		

### Dispatch(*Program*)

Argument	Type	Description
Program	Program Name	The name of the program to dispatch.

### Overview

This function runs the specified program in the background.

### Description

This function adds the specified program to a queue of programs to be run by a low priority task within the EDICT-97 runtime software. If the queue is full, the function will pause until a slot becomes available. Only one reference to a program can be in the queue at any one time, so calling the function twice will have no further effect until the program has been executed. The program is removed from the queue just before it is run, so a program can re-dispatch itself to keep it running continuously at a very low priority. Note that there is little guarantee as to how much processor time a program run with Dispatch will receive, and a complex program could take some time to execute.

### Return Type

This function does not return a value.

### Examples

Event	Enable	Action
All Input Complete	Default	Dispatch(FindAverage)
This would run the program "FindAverage" after all data entry was complete. The Dispatch function can be interrupted by other inputs.		

### Fill(*Dest, Value, Count*)

Argument	Type	Description
Destination	Data Reference	First item of destination data.
Value	Data Value	Data value to be used for fill.
Count	16-bit Unsigned	Number of data items to copy.

### Overview

This function sets *Count* data items from *Destination* onwards to *Value*.

### Description

This function sets data items as described above, where *Destination* is a reference to array elements of the same basic type as *Value*. This implies that both must refer to 16-bit integers, 32-bit integers, floating-point values or strings. The signed or unsigned attributes of the data do not matter, nor does the length of the strings.

### Return Type

This function does not return a value.

### Examples

Event	Enable	Action
Page Removed	Default	Fill(A[0], 0, 100)

This will load a value of zero to all locations A[0] through A[99] when another page is selected.

### FlipEntrySign()

Argument	Type	Description
None		

### Overview

This function flips the sign of the current data entry field.

### Description

EDICT-97 allows signed data entry, but not all operator terminals supported by the software are fitted with a key suitable for entering negative numbers. By assigning this function to a suitable key, you can overcome this limitation. You may choose to use a function key and make an entry in the Global Event Map, or you may prefer to designate a soft-key for this function, and make an appropriate entry in the page's event map. If data entry is not taking place, this function returns without action. Note: The Data Entry Property of the Data Entry field you are working with must allow negative and positive numbers to be entered.

### Return Type

This function does not return a value.

### Examples

Event	Enable	Action
Soft-Key 1 Pressed	Default	FlipEntrySign()

This will change the sign of the currently highlighted Data-Entry field.

### Format(*Template*, *Value*)

Argument	Type	Description
Template	String	The formatting template.
Value	32-bit Signed	The value to be formatted.

### Overview

This function formats the given number according to a formatting template.

### Description

This function expands the given number into a string, using the formatting template to decide upon the number of decimal places, the number base, and any units or other formatting which may be included. The function is often used with the General Text animation item and the CallString function to implement custom animation items.

### Return Type

This function returns a string.

### Examples

Event	Enable	Action
F1 Pressed	Default	Text := Format("0000.00", A[0])
F1 Released	Default	PrintMessage(2, Text)

The above two statements will convert the numerical value in A[0] into a string that will be printed to port 2. E.g., if A[0] equals 123456, the printout will show 1234.56.

### GetMessage(*Index*)

Argument	Type	Description
Index	Message Number	The number of the global message to display.

### Overview

This function selects the text of a given global message.

### Description

This function permits the user to access the global message table strings.

### Return Type

This function returns a string.

### Examples

Event	Enable	Action
F1 Pressed	Default	OutputString:=Left(GetMessage(Msg),10)

This would copy the 10 leftmost characters of the message "msg" to OutputString.

### GetNow()

Argument	Type	Description
None		

### Overview

This function gets the number of seconds from the beginning of the universe ( according to Paradigm ).

### Description

GetNow() returns a 32 bit unsigned number. This number can be used in functions such as GetYear, GetMonth, GetDate, GetDays, GetDay, GetWeek, GetWeekYear, GetHour, GetMin, GetSec, or in start/stop timer functions. For example, a user variable ThisMonth := GetMonth( GetNow() ) will put in ThisMonth the number of the month of the year.

### Return Type

This function returns a 32 bit unsigned integer.

### Examples

#### TRIGGER TABLE

Expression	Edge	Action
A[0].2 == TRUE	Rising	StartTime := GetNow()
A[0].2 == FALSE	Rising	ElapsedTime := GetNow() - StartTime

The two statements, above, detect when the third bit of register A[0] changes from false to true to false. On the first edge, the user variable StartTime is set to the current time. When the bit turns off, the user variable ElapsedTime is loaded with the number of seconds that had passed since the bit went true.

## GetTimer()

Argument	Type	Description
None		

### Overview

This function returns the value of the current task's timer.

### Description

Each task within the EDICT-97 runtime software has a timer associated with it. This timer can be loaded with a millisecond value, and it will then count down until it reaches zero. The timer can be read using this function, or set using the SetTimer function. These functions are generally employed within Roll-Your-Own-Protocol drivers to implement time-outs in Comms interactions. In other applications, you should not rely on the timer value being preserved once a given program has returned, as EDICT-97 may use the timer itself.

### Return Type

This function returns a 16-bit unsigned value.

### Examples

#### USER PROGRAM "CommsTask"

```
SetTimer(500);
```

```
while( GetTimer() ) Run(GetCommsReply);
```

The "while" statement will continually run the User Program "GetCommsReply" until the 500 millisecond timeout period expires.

## GotoField(Set)

Argument	Type	Description
Set	16 bit Unsigned	The number of the desired Data Entry field.

### Overview

This function selects a particular data entry field on a page.

### Description

This function permits positioning of the Data Entry Cursor on a particular field of the display. This can be useful if the programmer needs to provide a "Help" function. By saving the value of the current field, the operator can access a help screen, and return to the same data entry field. This number is 0 based, that is, the first data entry field on the page is 0.

### Return Type

This function returns 0 if Set is equal to or larger than the number of data entry fields.

### Examples

Event	Enable	Action
New Field Selected	Default	Field := Param
F1 Pressed	Field == 0	FieldSave := Field,GotoPage(Help0)
F1 Pressed	Field == 1	FieldSave := Field,GotoPage(Help1)
Page Selected	FieldSave <= 1	GotoField(FieldSave)

The above combination of Events, Enables, and Actions show one way of providing Help functions. The first line saves the number of the field as the operator uses NEXT or PREV to change fields. The second and third lines save that value, while changing to the Help pages. The fourth line verifies that the number in FieldSave is valid, and then positions the cursor on that field.

## GotoPage(*Page*)

Argument	Type	Description
Page	Page Name	The name of the page to display.

### Overview

This function selects the specified page for display.

### Description

This function instructs the system that the display page specified by *Page* should be displayed. Note that the page is not displayed immediately, but at the point when the system has finished processing the current event. This implies that several calls to this function made in response to the same event will cause the last referenced page to be displayed. When the new page is selected, a "Page Removed" event will be sent to the previous page, and a "Page Selected" event sent to the current page. If a further GotoPage function is called in response to the "Page Selected" event, the system will switch to that page without displaying the originally specified page. It is possible to lock the system by using this "ripple" technique to send it back-and-forth between two pages, so be careful to avoid such loops. Calling GotoPage in response to the "Page Removed" event will produce undefined behavior and should be avoided. A call to GotoPage will not have any effect at all if that page is already displayed.

### Return Type

This function does not return a value.

### Examples

Event	Enable	Action
Exit Key Pressed	Default	GotoPage(Page1)
When the Exit Key is pressed, this will display the page whose name is "Page1".		

## GotoPrevious()

Argument	Type	Description
None		

### Overview

This function returns to the previously displayed page.

### Description

This function "pops" a page name from an internal list of the last 20 pages displayed, and performs a GotoPage function to select that page. If the page contains data entry fields, the last active field will be re-selected. If you choose to use this function to back-up a menu structure, be sure to use it throughout. If you decide to use an explicit GotoPage call to return from one level, and then try to use GotoPrevious from the next higher level, the latter call will actually take you back down the tree, as the GotoPage call will have added the lower-level page to the page list. The moral is that you should use explicit GotoPage calls or GotoPrevious calls, but not a mixture of the two.

### Return Type

This function does not return a value.

### Examples

Event	Enable	Action
PREV key pressed	Default	GotoPrevious()
Returns to the previous page that was selected.		

## HexVal(*Text*)

Argument	Type	Description
Text	String	The string to be converted.

### Overview

This function converts a hex string into a numeric long.

### Description

This function converts its argument into a numeric value, taking the string to be a sequence of hex digits. It will stop processing the string at the first digit, which cannot be considered valid. The case of any alpha characters is ignored in any conversion.

### Return Type

This function returns a 32-bit unsigned value.

### Examples

Event	Enable	Action
F1 Pressed	Default	A[0] := HexVal("ABCD")

Pressing F1 will convert the ASCII characters ABCD into a 32 bit value, in this case the result = 43981. If the string were "123Z" the result would be the equivalent of hex 123 or decimal 291. The 291 will be 32 bits.

## HideMenu()

Argument	Type	Description
None		

### Overview

This function will cause a displayed Soft-Key Menu to be cleared.

### Description

When a Soft-key menu is configured and then displayed, the function HideMenu() will clear the text and disable the actions associated with the soft keys for that menu.

### Return Type

This function does not return a value.

### Examples

Event	Enable	Action
F1 Pressed	Default	HideMenu( )

Provided a Soft Key Menu is programmed and displayed, the text for that menu is blanked. Any action assigned to the Soft Keys for that menu are now disabled. **See also:** ShowMenu(), IsMenuActive(), ToggleMenu().

## HoldTx(*Port, State*)

Argument	Type	Description
Port	16-bit Unsigned	The required serial port.
State	16-bit Unsigned	The transmission hold flag.

### Overview

This function holds or releases transmission on the indicated serial port.

### Description

This function should only be called for a port to which the Roll-Your-Own Protocol driver has been bound. If called with a *true* value for the *State* argument, transmission on the indicated port will be disabled, and any subsequent calls to functions which would normally send data will result in that data being buffered in memory. When the function is then called with a *false* value for the *State* argument, transmission will be re-enabled and the data will be released from the buffer. The function is often used to allow a Comms frame to be assembled in memory and then transmitted in one single burst. If the function is not used, a task switch could cause gaps in the data stream.

### Return Type

This function does not return a value.

### Examples

Event	Enable	Action
F1 Pressed	Default	HoldTx(3, TRUE);

Disables the transmission of any data out of port 3 until the TRUE is changed to a value equal to FALSE, or 0.

### IsMenuActive()

Argument	Type	Description
None		

### Overview

Indicates if a Soft Key Menu is displayed.

### Description

This function returns a non-zero value if a Soft Key Menu is displayed, 0 otherwise.

### Return Type

This function returns an Unsigned 16 bit value.

### Examples

Event	Enable	Action
Soft Key 1 Pressed	IsMenuActive() == 0	Run(Program1)

Runs Program1 if the Menu is not displayed. Otherwise, it will perform the action specified for the menu.

**See also:** ShowMenu(), HideMenu(), ToggleMenu().

### IsOnLine(*Port*)

Argument	Type	Description
Port	16-bit Unsigned	The required serial port.

### Overview

This function indicates if the specified port is currently on-line.

### Description

This function can be used when a link driver has been specified for a serial port. It returns a value of *true* or *false*, indicating if the link is currently on-line. If no link driver has been specified, a value of *true* will always be returned. This function can be used within the Trigger Table to take certain actions when a port goes on-line or off-line, although a similar result can be obtained by responding to the events associated with that port.

### Return Type

This function returns a 16-bit unsigned value.

### Examples

Event	Enable	Action
-------	--------	--------

Comms Update	Default	A[0] := IsOnLine(1)
--------------	---------	---------------------

A[0] will equal 0 if the port is specified and not communicating. A[0] will be some non-zero value otherwise.

### Left(*Text*, *Count*)

Argument	Type	Description
Text	String	The string to be processed.
Count	16-bit Unsigned	The number of characters to extract.

### Overview

This function returns the left-most *Count* characters of *Text*.

### Description

This function extracts a sub-string from *Text*, starting at the first position and extending for *Count* characters. If *Count* exceeds the length of the string, the entire string is returned, without the addition of any padding.

### Return Type

This function returns a string.

### Examples

Event	Enable	Action
Soft-key 1 pressed	default	Data := Left("Hello World", 5)
Soft-key 2 pressed	default	Data := Left("Hi Mom!", 4)

The string variable "Data" will be equal to 'Hello', if Soft-Key 1 is pressed.

"Data" will be equal to 'Hi M' if Soft-Key 2 is pressed.

### Len(*Text*)

Argument	Type	Description
Text	String	The string to be processed.

### Overview

This function returns the number of characters in *Text*.

### Description

This function returns the number of characters in its argument.

### Return Type

This function returns a 16-bit unsigned value.

### Examples

Event	Enable	Action
F1 Pressed	Default	Data := Len("Fred")

Data, a numeric variable, will be equal to 4.

## **LogClear(Log)**

## **LogDump(Log, Port, Count)**

## **LogEvent(Log, Code, Param)**

<b>Argument</b>	<b>Type</b>	<b>Description</b>
Log	Log Name	The name of the log to be accessed.
Port	16 bit unsigned	Number of the port.
Count	16 bit unsigned	Number of items to print.
Code	16 bit unsigned	Number of the event.
Param	32 bit unsigned	Optional Parameter for the Event.

### **Overview**

These function provide support for the Event Logger function in the Graphical units.

### **Description**

Somewhat self-descriptive, these functions permit user control of the logging of events. LogClear() takes, as an argument, the name of a log configured in Event Logs. LogDump() prints Count number of items, from the specified log to the specified port. LogEvent() adds event number "Code" to the specified log. An optional parameter can be logged by including a numerical template in curly brackets within the text to be logged. For example, in Event Logs, log entry 7 in the log named Welding might say "Machine Stopped { 000 }". Executing the function LogEvent( Welding, 7, 4 ) will cause 4 to replace the {000} in the display of the log.

### **Return Type**

None of the functions return a value.

### **Examples**

<b>Event</b>	<b>Enable</b>	<b>Action</b>
F1 Pressed	Default	LogDump( Welding, 2, 100 )
This function prints the first 100 entries of the log named Welding, to port 2.		
F2 Pressed	Default	LogClear( Welding )
This function clears the log named Welding..		

## **LogOff()**

<b>Argument</b>	<b>Type</b>	<b>Description</b>
None		

### **Overview**

This function logs off the current user and resets the security level.

### **Description**

Executing this function when user 14 is logged on, removes user 14, and inactivates any function that requires secure access.

### **Return Type**

This function returns nothing.

### **Examples**

<b>Event</b>	<b>Enable</b>	<b>Action</b>
EXIT Key Pressed	Default	LogOff()
This removes the current user from the system, and will disable access to any function requiring security permission.		

## MakeChar(Value)

Argument	Type	Description
Value	16-bit Signed	The value to be converted.

### Overview

This function converts a numeric value to a single character string.

### Description

The function returns a single character string containing a character with the same ASCII value as the function's argument. If a value of zero is passed, an empty string will be returned. This function is often used to assemble a string from Comms data or other sources before extracting values using the Val or HexVal functions. It can also be used to build a string when simulating alphanumeric input.

### Return Type

This function returns a string.

### Examples

#### USER PROGRAM

```
...
Text := "ABC";
Data := 3;
Text += MakeChar(Data);
Data := 4;
Text += MakeChar(Data);
...
```

"Text" will now be 'ABC34'.

## Max(Value1, Value2)

Argument	Type	Description
Value1	Numeric	The 1 <sup>st</sup> value to be processed.
Value2	Numeric	The 2 <sup>nd</sup> value to be processed.

### Overview

This function returns the larger of its two arguments.

### Description

This function returns the larger of its two arguments, taking into consideration the data type of each. If an unsigned value and a signed value are passed, EDICT-97's behavior is undefined if the signed value should be negative. You should thus be careful to pass values of similar type if you are dealing with signed numbers.

### Return Type

This function returns a value of the type required to hold either argument.

### Examples

Event	Enable	Action
Comms Update	Default	A[0] := Max(A[1], A[2])

A[0] is now the larger of A[1] and A[2].

## Mean(*Data*, *Count*)

Argument	Type	Description
Data	Data Reference	First item of the data array.
Count	16-bit Unsigned	The number of data points.

### Overview

This function returns the mean value of an array of data points.

### Description

This function returns the mean value of the data points indicated by its arguments. If the *Count* argument is zero, a value of zero will always be returned. The internal computations are carried out using floating point maths, irrespective of the data type of the underlying data. This ensures that no overflows can occur, even with a large number of large data values. If you need to use the resulting value where only an integer is permitted, you should use a type cast sequence to perform the type conversion, optionally multiplying by a power of ten first to maintain a level of decimal precision.

### Return Type

This function returns a floating point value.

### Examples

Event	Enable	Action
F1 Pressed	Default	Average := Mean(A[0], 10)

“Average” is now the average of the 10 points A[0] through A[9].

## Mid(*Text*, *Offset*, *Count*)

Argument	Type	Description
Text	String	The string to be processed.
Offset	16-bit Unsigned	The zero-based starting position.
Count	16-bit Unsigned	The number of characters to extract.

### Overview

This function returns *Count* characters from position *Offset* in *Text*.

### Description

This function extracts a sub-string from *Text*, starting at the position *Offset* and extending for *Count* characters. If *Count* plus *Offset* exceeds the length of the string, the entire string is returned, without the addition of any padding. You may pass a value of -1 for *Count* to indicate that you want the whole of the rest of the string to be returned.

### Return Type

This function returns a string.

### Examples

Event	Enable	Action
F1 Pressed	Default	Data1 := Mid(“Hello EDICT-97 World”, 6, 5)
F1 Pressed	Default	Data2 := Mid(“Hello Fred”, 6, 5)
F1 Pressed	Default	Data3 := Mid(“Hello World & Fred”, 8,-1)

Data1 equals ‘EDICT-97’

Data2 equals ‘Fred’

Data3 equals ‘rld & Fred’

Another explanation for *Offset* is that it is equal to the number of characters ignored.

### **Min(Value1, Value2)**

<b>Argument</b>	<b>Type</b>	<b>Description</b>
Value1	Numeric	The 1 <sup>st</sup> value to be processed.
Value2	Numeric	The 2 <sup>nd</sup> value to be processed.

#### **Overview**

This function returns the smaller of its two arguments.

#### **Description**

This function returns the smaller of its two arguments, taking into consideration the data type of each. If an unsigned value and a signed value are passed, EDICT-97's behavior is undefined if the signed value should be negative. You should thus be careful to pass values of similar type if you are dealing with signed numbers.

#### **Return Type**

This function returns a value of the same required to hold either argument.

#### **Examples**

<b>Event</b>	<b>Enable</b>	<b>Action</b>
Comms Update	Default	A[0] := Min(A[1], A[2])
A[0] is the smaller of A[1] and A[2].		

### **ModemAnswer(Port)**

### **ModemDial(Port, Number)**

### **ModemHangUp(Port)**

### **ModemRinging(Port)**

<b>Argument</b>	<b>Type</b>	<b>Description</b>
Port	Numeric	The number of the port for the modem.
Number	Numeric	The phone number.

#### **Overview**

These functions provide modem support.

#### **Description**

ModemDial(), ModemAnswer(), and ModemHangup() are Actions that can be performed in response to Events. ModemRinging() is a function that returns TRUE if the modem is ringing. One would usually put this function in the enable field with action ModemAnswer(). Once the connection is established, serial communications can proceed normally through the selected port.

#### **Return Type**

ModemHangUp() returns nothing. ModemRinging returns TRUE if the Ring signal is being received. ModemDial, and ModemAnswer return Hayes standard codes:

```
OK      0
RC_CONNECT  1
RC_RING   2
RC_NO_CARRIER  3
RC_ERROR   4
RC_NO_DIAL  6
RC_BUSY    7
RC_NO_REPLY  8
```

or any other code returned by the specific modem.

## Examples

Event	Enable	Action
F1 Pressed	Default	ModemDial( 2, "8885551234")
Comms Update	ModemRing(2)	ModemAnswer(2)
F2 Pressed	Default	ModemHangUp(2)

## MuteSiren()

Argument	Type	Description
None		

## Overview

This function turns off the system's internal sounder.

## Description

This function turns off the system's internal sounder, which may have been previously activated by a call to SirenOn, or by the activation of a suitable alarm. Note that this function also clears the IsSirenOn system variable, which tracks the state of the siren.

## Return Type

This function does not return a value.

## Examples

Event	Enable	Action
MUTE Key Pressed	Default	MuteSiren()

This statement, normally put in GLOBAL EVENTS, will turn off the siren.

## PopDev(Data, Count)

Argument	Type	Description
Data	Data Reference	First item of the data array.
Count	16-bit Unsigned	The number of data points.

## Overview

This function returns the standard deviation of an array of data points, assuming the data points to represent the whole of the population under study. If you need to find the standard deviation of a sample, please use the StdDev function instead.

## Description

This function returns the standard deviation of the data points indicated by its arguments. If the *Count* argument is zero, a value of zero will always be returned. The internal computations are carried out using floating point maths, irrespective of the data type of the underlying data. This ensures that no overflows can occur, even with a large number of large data values. If you need to use the resulting value where only an integer is permitted, you should use a type cast sequence to perform the type conversion, optionally multiplying by a power of ten first to maintain decimal precision.

## Return Type

This function returns a floating point value.

## Examples

Event	Enable	Action
Comms Update	BatchDone	Dev := PopDev(A[0], 10)

Calculates, when "BatchDone" is not zero, the standard deviation for all 10 items in the batch.

## PostEvent(*Param*)

Argument	Type	Description
Parameter	32-bit Signed	The event parameter.

### Overview

This function posts a user event to the system event queue.

### Description

This function stores an event of type "User Generated Event" in the system event queue, with the event parameters being set to the specified value. When the event is being processed, this parameter will be placed in the Param system variable, and can be used in any actions that handle the event. The function is used by advanced EDICT-97 users who might want some activity to pass an event to the user interface system. As an example, you might create Trigger Table entries to post user generated events when a number of bits change in the PLC, with these bits being "connected" to push buttons on a remote control panel. By placing suitable entries in an event map and using the Enable column to compare Param to the a unique value specified for each bit, you can respond to these remote push buttons as you would keys on the terminal's own keyboard. Another use of this function is to provide a free-running chain of events for demonstration databases and the like. The technique involves posting an event when the page in question is first selected, and using the event to post another event and to perform some processing. The effect will be to execute this processing at the maximum rate the system can maintain, irrespective of any Comms updates and the like. This technique places a high load on the system and should not be used in real-life applications.

### Return Type

This function does not return a value.

### Examples

#### Trigger Table

Expression	Edge	Action
A[0] && 0x4	Rising	PostUserEvent(0x04)

This will set Param to 4. Then Param could be used to perform an action, say:

Event	Enable	Action
Comms Update	Param == 4	Gotopage(Page7)

Where Page7 instructs the operator to perform some action appropriate for that particular input bit that was set.

## PrintFormFeed(*Port*)

Argument	Type	Description
Port	16-bit Unsigned	The target port, or zero for default.

### Overview

This function sends a form-feed to the indicated printer.

### Description

This function sends a form-feed sequence to the printer indicated by *Port*, or to the first configured printer driver if a value of zero is passed. If the Comms port in question does not have a printer driver configured in the Comms Drivers table, no action will result from calling this function. Note that some printer drivers may choose to ignore form feed requests if they do not make sense in the context of the driver in question. This function will return immediately if memory is available to queue the printer request, but may take some time to return if the printer is very busy.

### Return Type

This function does not return a value.

## Examples

Event	Enable	Action
F8 Pressed	Default	PrintFormFeed(1)
F8 Released	Default	PrintReport(1,Report1)

Pressing F8 with the above sequence will form-feed the printer in preparation for "Report1" to be printed on Port 1.

## PrintMessage(*Port, Message*)

Argument	Type	Description
Port	16-bit Unsigned	The target port, or zero for default.
Message	16-bit Unsigned	The message number to print.

### Overview

This function printed the indicated message to the indicated printer.

### Description

This function sends the text of the Message Table entry indicated by *Message* to the printer indicated by *Port*, or to the first configured printer driver if a value of zero is passed. If the Comms port in question does not have a printer driver configured in the Comms Drivers table, no action will result from calling this function. The message text will not be followed by a carriage return sequence, so as to allow you to combine messages and other printer primitives on the same line. This function will return immediately if memory is available to queue the printer request, but may take some time to return if the printer is very busy.

### Return Type

This function does not return a value.

## Examples

Event	Enable	Action
Soft-Key 1 Pressed	Default	PrintMessage(2, 10)

This will print Message 10 from the Global Message Table to Port 2.

## PrintNewLine(*Port*)

Argument	Type	Description
Port	16-bit Unsigned	The target port, or zero for default.

### Overview

This function sends a new-line sequence to the indicated printer.

### Description

This function sends a new-line sequence to the printer indicated by *Port*, or to the first configured printer driver if a value of zero is passed. If the Comms port in question does not have a printer driver configured in the Comms Drivers table, no action will result from calling this function. This function will return immediately if memory is available to queue the printer request, but may take some time to return if the printer is very busy.

### Return Type

This function does not return a value.

## Examples

Event	Enable	Action
Soft-Key 1 Pressed	Default	PrintLineFeed(2)

This will send a Line Feed to the printer connected to port 2.

## PrintReport(*Port*, *Report*)

Argument	Type	Description
Port	16-bit Unsigned	The target port, or zero for default.
Report	Report Name	The report to be printed.

### Overview

This function prints the specified report to the indicated printer.

### Description

This function sends the report indicated by *Report* to the printer indicated by *Port*, or to the first configured printer driver if a value of zero is passed. If the Comms port in question does not have a printer driver configured in the Comms Drivers table, no action will result from calling this function. This function will return immediately if memory is available to queue the printer request, but may take some time to return if the printer is very busy. Any animation items on the report will be expanded before the function returns, so subsequent changes to the underlying data will not affect the print-out.

### Return Type

This function does not return a value.

### Examples

#### Schedule Table

Days	Hour	Min	Sec	Action
Weekdays	17	00	00	PrintReport(0, BatchReport)

This will print the report named "BatchReport" at 5:00 PM Mondays through Fridays on the first port found that is selected as a printer driver.

## PrintString(*Port*, *String*)

Argument	Type	Description
Port	16-bit Unsigned	The target port, or zero for default.
String	Character String	The string to be printed.

### Overview

This function prints the indicated string to the indicated printer.

### Description

This function sends *String* to the printer indicated by *Port*, or to the first configured printer driver if a value of zero is passed. If the Comms port in question does not have a printer driver configured in the Comms Drivers table, no action will result from calling this function. This function will return immediately if memory is available to queue the printer request, but may take some time to return if the printer is very busy. Control characters in the string may or may not be honored, according to the printer driver in question, but all drivers will treat the "\n" character as a new line sequence.

### Return Type

This function does not return a value.

### Examples

Event	Enable	Action
F1 Pressed	Default	PrintString(0, "Hello World!\n")
F2 Pressed	Default	PrintString(0, MyName)

Pressing F1 will print Hello World! with a carriage return and a line feed. Pressing F2 will print the string stored in the variable MyName, without adding any other characters. Both will go to the lowest numbered port that is assigned a printer driver

## PrintTimeStamp(*Port*)

Argument	Type	Description
Port	16-bit Unsigned	The target port, or zero for default.

### Overview

This function prints a time-and-date stamp to the indicated port.

### Description

This function sends a time-and-date stamp sequence to the printer indicated by *Port*, or to the first configured printer driver if a value of zero is passed. If the Comms port in question does not have a printer driver configured in the Comms Drivers table, no action will result from calling this function. This function will return immediately if memory is available to queue the printer request, but may take some time to return if the printer is very busy. The time-and-date stamp will be formatted according to the language selected in the database's terminal properties, and will reflect the time at which the request was queued, rather than the time at which it was printed.

### Return Type

This function does not return a value.

### Examples

Event	Enable	Action
Comms Update	A[0]>=50	PrintTimeStamp(1)

This will print the time and date when Comms Block location A[0] is greater than 49. This could be followed by a PrintMessage command to show when an event occurred, and what happened.

## Random(*Range*)

Argument	Type	Description
Range	16-bit Unsigned	The range of numbers to return.

### Overview

This function returns a random number from 0 to *Range* - 1.

### Description

This function uses EDICT-97's internal random number generator to produce a pseudo-random number in the range required. The distribution of numbers produced should be more or less uniform, although true randomness can never be achieved by purely software means. EDICT-97 continually evaluates random numbers in the background whenever it has spare processing time, so the sequence of numbers generated by this function will be different each time EDICT-97 starts execution.

### Return Type

This function returns a 16-bit unsigned value.

### Examples

Event	Enable	Action
F1 Pressed	Default	A[0] := Random(100)

Generates a pseudo-random number between 0 and 99.

## ReadBlock(*Block*)

Argument	Type	Description
Block	Block Name	The Comms block to be read.

## Overview

This function indicates that the indicated block should be read from the PLC.

## Description

This function instructs the Comms task to read the indicated communications block from the associated device on the next Comms scan. The function returns without waiting for the Comms update to be completed, so you should not assume that the data will be in memory as soon as control returns. Neither should you assume that calling the function for two blocks will read the data in the order the functions were called. The ReadBlock function can be called for blocks of any access type, overriding whatever settings are made in the Comms Blocks table.

## Return Type

This function does not return a value.

## Examples

Event	Enable	Action
NEXT Key Pressed	Default	ReadBlock(Q)

This will read comms block Q, even if it is defined as a write block, on the comms scan following the pressing of the NEXT key.

## RelayClose()

## RelayOpen()

Argument	Type	Description
None		

## Overview

These functions manipulate the relay on units so equipped.

## Return Type

This function does not return a value.

## Examples

Event	Enable	Action
Alarm Activated	Default	RelayClose()
Alarm Accepted	Default	RelayOpen()

On an Alarm condition, closes the relay until the alarm has been acknowledged.

## Right(*Text*, *Count*)

Argument	Type	Description
Text	String	The string to be processed.
Count	16-bit Unsigned	The number of characters to extract.

## Overview

This function returns the right-most *Count* characters of *Text*.

## Description

This function extracts a sub-string from *Text*, starting at the position *Count* from the end of the string. If *Count* plus exceeds the length of the string, the entire string is returned, without the addition of any padding.

### Return Type

This function returns a string.

### Examples

Event	Enable	Action
F1 Pressed	Default	Data1 := Right("Hello World", 5)
F1 Pressed	Default	Data2 := Right("Fred", 8)
Data1 now equals 'World'.		
Data2 now equals 'Fred'.		

### Run(*Program*)

Argument	Type	Description
Program	Program Name	The name of the program to run.

### Overview

This function runs the program indicated.

### Description

This function runs the program indicated, and only returns when the last statement of the program has been executed, or when a "return" statement is encountered. Any value passed to the return statement is discarded. A program can call another program up to a depth of about fifteen levels, and a program can even call itself. If you nest program calls too deeply, you will get a stack error, which will trip the system watchdog. This should not prove a problem, however, as nesting to a level sufficient to cause this kind of error is virtually unheard of.

### Return Type

This function does not return a value.

### Examples

Event	Enable	Action
All Input Complete	Default	Run(FindTotals)

This will execute the USER PROGRAM "FindTotals" after all data entry fields have been completed. The terminal will not process any other inputs until the USER PROGRAM is completed.

### SerialPrint(*Port*, *Text*)

Argument	Type	Description
Port	16-bit Unsigned	The required serial port.
Text	String	The data to be transmitted.

### Overview

This function sends a character string to the given serial port.

### Description

This function should only be called for a port to which the Roll-Your-Own Protocol driver has been bound. It is used to transmit a sequence of characters to the port, and can be used to avoid repeated calls to the SerialWrite function. If you call this function after disabling transmission on the port by using the HoldTx function, ensure that you do not exceed the default buffer size of 256 character. If you attempt to store more characters than this while transmission is disabled, the results are undefined.

### Return Type

This function does not return a value.

### Examples

Event	Enable	Action
F1 Pressed	Default	SerialPrint(1, "DRD" + Format("000", Reg))

When F1 is pressed, the terminal will take the value in the variable called Reg, convert the last 3 digits to 3 ASCII digits, append them to the string DRD, and prints that to port 1. (If Reg equals 123, the printout is DRD123).

### SerialRead(*Port*)

Argument	Type	Description
Port	16-bit Unsigned	The required serial port.

### Overview

This function reads a character from the indicated serial port.

### Description

This function should only be called for a port to which the Roll-Your-Own Protocol driver has been bound. It returns the next character stored in the port's buffer, or a value of 0xFFFF is no character is available. While waiting for characters in your receive routine, you should ideally call the Sleep function to yield the processor if no data is available. Failure to do this will adversely effect other areas of the runtime software.

### Return Type

This function returns a 16-bit unsigned value.

### Examples

#### USER PROGRAMS

```
Data := SerialRead(1);
if ( Data == 0xFFFF )
    return;
else .....
```

The above lines in a user program will check serial port 1 and if there are no characters waiting, will return. Otherwise, it will continue the program.

### SerialWrite(*Port, Data*)

Argument	Type	Description
Port	16-bit Unsigned	The required serial port.
Data	16-bit Unsigned	The byte to be transmitted.

### Overview

This function transmits a given byte on the indicated serial port.

### Description

This function should only be called for a port to which the Roll-Your-Own Protocol driver has been bound. It transmits the given byte of data using the indicated port, or places it in the appropriate memory buffer if the port is

busy or if the HoldTx function has been used to delay transmission. The return value indicates whether or not the function has been successful. A return value of *false* indicates that no room was available in the buffer to hold the data, and that you should call the function again after calling the Sleep function to yield the processor for a short period of time.

If you are using SerialWrite with an RS-485 port, you will need to turn on the RTS signal in order to enable the port's transmitter. This is done using the SetRTS function, but you need to take care to make sure that you do not turn the signal off too early, and so "chop" the frame. The best way to do this is to use the UseAutoEnables function to put the port into auto-enables mode, as the hardware will then look after the timing issues for you. Refer to this function for more information on this subject.

### Return Type

This function returns a 16-bit unsigned value.

### Examples

#### USER PROGRAM

...

```
SerialWrite(1, 0x10);
```

```
SerialWrite(1, 0x32);
```

...

The above lines write out a line feed (0x10), and an ASCII 2 (0x32).

### SetBreak(*Port*, *State*)

Argument	Type	Description
Port	16-bit Unsigned	The required serial port.
State	16-bit Unsigned	The required "break" state.

### Overview

This function places the indicated port in the "break" state or cancels that state.

### Description

This function should only be called for a port to which the Roll-Your-Own Protocol driver has been bound. When called with a *State* argument of *true*, the indicated port will be put into the "break" state. This state will be cancelled if the function is called with a *State* argument of *false*. Most protocols, which use the "break" state for signaling, expect the state to exist for a certain amount of time, so it is usual to call the Sleep function between related calls to SetBreak. Note that any timing will always be approximate.

### Return Type

This function does not return a value.

### Examples

#### USER PROGRAM

...

```
SetBreak(1, TRUE);
```

```
Sleep(100);
```

```
SetBreak(1, FALSE);
```

...

The above line will create a 100 millisecond 'break' condition on the transmit line to signal the receiving device.

## SetCommsTask(*Port, Program*)

Argument	Type	Description
Port	16-bit Unsigned	The required serial port.
Program	Program Name	The program to execute.

### Overview

This function binds a given program to the Comms task for a given serial port.

### Description

This function indicates that the system should execute the specified program after every update on the specified serial port. The program will be called in the context of the Comms task responsible for the port in question, and not be the user interface task. You should thus be careful when accessing related data items as you may find that another task modifies these items unexpectedly. The function is typically to assign a program to carry out custom Comms processing when using the Roll-Your-Own-Protocol driver.

### Return Type

This function does not return a value.

### Examples

#### GLOBAL EVENTS

Event	Enable	Action
System Initialized	Default	SetCommsTask(1, CommsTask)

This will set up, for communications, the program "CommsTask". This will cause that program to run when a communications scan occurs.

## SetLanguage(*Language*)

Argument	Type	Description
Language	16-bit Unsigned	The desired language number.

### Overview

This function selects the desired language for text animation items.

### Description

This function can be called to select a configured language to be used for text animation items, such as Status Text, General Text, etc. Provided additional languages are configured in File/Database Information, this function permits the operator to change the language to be used. Seven language titles are defined, UK English, US English, French, German, Italian, Spanish, and Custom. The programmer assigns the language associated with the Language argument in File/Database Information, and then selects File/Translations to assign the strings.

### Return Type

This function does not return a value.

### Examples

#### DISPLAY PAGE-Data Entry Field with value = Language

Event	Enable	Action
Soft Key 1 Pressed	Language<=3	SetLanguage(Language++)
Soft Key 1 Pressed	Language>3	Language := 0,SetLanguage(0)

The above will permit the selection of up to 4 languages assuming they are programmed 0

## SetRTS(*Port*, *State*)

Argument	Type	Description
Port	16-bit Unsigned	The required serial port.
State	16-bit Unsigned	The required RTS state.

### Overview

This function sets the state of the RTS control line on the indicated serial port.

### Description

This function should only be called for a port to which the Roll-Your-Own Protocol driver has been bound. The RTS line on the indicated port will be asserted if the *State* argument is *true*, and de-asserted if the argument is *false*. If the port has been placed into “auto-enables” mode, RTS will not be allowed to return to a de-asserted state until all the characters in the transmit queue have cleared the serial port hardware. As the RTS line is used to key the transmitter on RS-485 ports, this technique is often used to ensure that the port is trio-stated at exactly the point required to allow a frame to be sent.

### Return Type

This function does not return a value.

### Examples

#### USER PROGRAM

```
...
SetRTS(3, TRUE)
SerialWrite(3, 0x32)
SetRTS(3, FALSE)
...
```

This will enable the transmission of the hex value 32 from the RS485 Serial Port.

## SetTimer(*Value*)

Argument	Type	Description
Value	16-bit Unsigned	The timer value in milliseconds.

### Overview

This function sets the current task's timer to the indicated value.

### Description

Each task within the EDICT-97 runtime software has a timer associated with it. This timer can be loaded with a millisecond value, and it will then count down until it reaches zero. The timer can be set using this function, or read using the GetTimer function. These functions are generally employed within Roll-Your-Own-Protocol drivers to implement time-outs in Comms interactions. In other applications, you should not rely on the timer value being preserved once a given program has returned, as EDICT-97 may use the timer itself.

### Return Type

This function does not return a value.

### Examples

#### USER PROGRAM

```
...
SetTimer(500)
...
```

This will set a 500 millisecond timer, allowing the subsequent code to take appropriate action at the end of the time period.

## Sgn(Value)

Argument	Type	Description
Value	Numeric	The value to be processed.

### Overview

This function returns the sign of its argument.

### Description

If the argument to this function is negative, a value of minus one will be returned; if the argument is positive, a value of plus one will be returned; if the argument is zero, a result of zero will be returned. We can thus state that  $X$  is equal to  $\text{Sgn}(X) \cdot \text{Abs}(X)$  for all  $X$ .

### Return Type

This function returns a value of the same type as its argument.

### Examples

Event	Enable	Action
F1 Pressed	Default	$A[0] := \text{Sgn}(A[1])$
If $A[1]$ is 0, $A[0]$ is 0.		
If $A[1]$ is positive, $A[0]$ is 1.		
If $A[1]$ is negative, $A[0]$ is -1. The actual value stored in memory depends upon the type.		

## ShowMenu(Menu)

Argument	Type	Description
Menu	Menu Name	Menu Name assigned in Soft Key Menus.

### Overview

This function displays the menu corresponding to the name given, which, in turn, assigns functions to the associated soft keys.

### Description

Once Soft Key Menus have been configured in the appropriate section of DataBase Contents, one selects which menu is to be displayed according to the name given to that menu. ShowMenu(Menu1), will display Menu1 and display the programmed text next to the Soft Keys, and create the linkage to the functions associated with those soft keys, such as Key Pressed, Key Released, etc.

### Return Type

This function does not return a value.

### Examples

Event	Enable	Action
Page Selected	Default	ShowMenu(Recipe1)
Comms Update	Error == TRUE	ShowMenu(Diagnostics1)

When the page is selected, Recipe1 Menu will display at the Soft Keys. If the user variable Error gets set, the Recipe1 menu will disappear, along with its assigned Soft Key functions, and the menu Diagnostics1 will appear, which can assign different functions to the Soft Keys. **See also** HideMenu(), IsMenuActive(), ToggleMenu()

## SirenOn()

Argument	Type	Description
None		

### Overview

This function turns on the system's internal sounder.

### Description

This function turns on the system's internal sounder, in the same way as it is activated by the triggering of a suitable alarm. Note that this function also sets the IsSirenOn system variable, which tracks the state of the siren.

### Return Type

This function does not return a value.

### Examples

Event	Enable	Action
Existing Link Broken	Default	SirenOn()

This will turn on the siren if the communications link is broken.

## Sleep(*Period*)

Argument	Type	Description
Period	16-bit Unsigned	The required period in milliseconds.

### Overview

This function suspends the current task for the specified period.

### Description

This function suspends the current task for the specified period. This advanced function should not be used unless you have an understanding of how EDICT-97's internal multitasking works, and of the implications of suspending a given task. Things it should not be used for include attempting to create timed pulses in the PLC, or pausing before switching away from a display page. The most common use of Sleep is to separate Beep commands, either within successive lines of an event map, or within a program. You can also use it to slow-down programs run in response to trigger table or schedule table entries, but we're getting into the realms of magic here so I shall say no more.

### Return Type

This function does not return a value.

### Examples

#### USER PROGRAM

....

Sleep(100)

....

This will pause the user program for 100 milliseconds.

## Sqrt(Value)

Argument	Type	Description
Value	Numeric	The value to be processed.

### Overview

This function returns the square root of its argument.

### Description

This function returns the square root of its argument. Passing a negative value will produce an undefined result, and may trip the terminal's watchdog. Note that the returned value is always a floating point value. If you need to use the resulting value where only an integer is permitted, you should use a type cast sequence to perform the type conversion, optionally multiplying by a power of ten first to maintain a level of decimal precision.

### Return Type

This function returns a floating point value.

### Examples

Event	Enable	Action
F1 Pressed	Default	A[0] := Sqrt(2)

A[0] is the Square Root of 2 (1.4142..).

## StdDev(Data, Count)

Argument	Type	Description
Data	Data Reference	First item of the data array.
Count	16-bit Unsigned	The number of data points.

### Overview

This function returns the standard deviation of an array of data points, assuming the data points to represent a sample of the population under study. If you need to find the standard deviation of the whole population, please use the PopDev function instead.

### Description

This function returns the standard deviation of the data points indicated by its arguments. If the *Count* argument is zero, a value of zero will always be returned. The internal computations are carried out using floating point maths, irrespective of the data type of the underlying data. This ensures that no overflows can occur, even with a large number of large data values. If you need to use the resulting value where only an integer is permitted, you should use a type cast sequence to perform the type conversion, optionally multiplying by a power of ten first to maintain a level of decimal precision.

### Return Type

This function returns a floating point value.

### Examples

Event	Enable	Action
Comms Update	BatchDone	Dev := StdDev(A[0], 10)

'Dev' will be equal to the standard deviation of A[0] through A[9], assuming that the entire batch is larger than 10.

## StopSystem()

Argument	Type	Description
None		

### Overview

This function stops the runtime system.

### Description

This function stops the runtime system, and allows the system to accept downloads from the configuration software. If the system is power-cycled while in this state, it will restart as this function does not clear the "database valid" flag. You will not normally have to call this function, as EDICT-97 will continue to monitor the programming port during execution, unless a Comms device has been assigned to that port. In these cases, you should ensure that you provide some way of running this function if you want to download to the terminal without having to clear its database. Hiding the function at the bottom of a menu structure somewhere is a good idea, as operators will otherwise take great pleasure in making mischief by stopping the system every time they get bored. You may even like to implement a system whereby a given key combination must be pressed before the function is called. This is best done by setting and clearing bits within an internal variable in response to key-pressed and key-release events, and having an entry in the Trigger Table which calls this function when a given bit pattern is present.

### Return Type

This function does not return a value.

### Examples

Event	Enable	Action
F8 Pressed	StopSystemEnabled==TRUE	StopSystem()

This will, if the variable "StopSystemEnabled" is TRUE, put the panel into "DOWNLOAD". This is necessary when Port 1 is assigned to any driver, otherwise, there is no way to reprogram, except by clearing the terminal memory ( pressing the MUTE and EXIT keys, simultaneously, while applying power).

## Time(Hour, Minute, Second)

Argument	Type	Description
Hour	16 bit unsigned	Number of the hour.
Minute	16 bit unsigned	Number of the minute.
Second	16 bit unsigned	Number of the second.

### Overview

This function generates a 32 bit number that represents the number of seconds from the beginning of the day.

### Description

Calling Time() after assigning values to Hour, Minute, and Second, will give a 32 bit unsigned number that represents the number of seconds from the beginning of the day to the time given. See also Date().

### Return Type

This function returns a 32 bit unsigned number.

### Examples

Event	Enable	Action
F1 Pressed	Default	TrendDefer( 1, Time( 22, 59, 00 ) )

The above line will put off trending on channel 1 until 10:59 PM.

## **ToggleMenu(Menu)**

<b>Argument</b>	<b>Type</b>	<b>Description</b>
Menu	Menu Name	Menu Name assigned in Soft Key Menus.

### **Overview**

This function causes the given menu to display and hide.

### **Description**

ToggleMenu() will alternately hide and display a given menu. The difference between it, and ShowMenu()/HideMenu(), is that the latter would require two separate events, whereas ToggleMenu will turn it on when off, and off when on.

### **Return Type**

This function does not return a value.

### **Examples**

<b>Event</b>	<b>Enable</b>	<b>Action</b>
F1 Pressed	Default	ToggleMenu(Menu3)

This will display Menu3 on if it is not currently being displayed, and hide it if it is being displayed.

**See also:** ShowMenu(), HideMenu, IsMenuActive()

## **TrendClear(Channel)**

## **TrendDefer(Channel, Time)**

## **TrendStart(Channel)**

## **TrendStop(Channel)**

<b>Argument</b>	<b>Type</b>	<b>Description</b>
Channel	Channel Number	Number of the Channel configured in Data Logger.
Time	32 bit unsigned	Time (in seconds).

### **Overview**

These functions control the operation of the Data Logger.

### **Description**

TrendClear(), TrendStart(), and TrendStop() are self-explanatory. The programmer need only specify the number of the channel ( selected in Data Logger ). TrendDefer() puts off the running of the trend until the specified time is reached. This time is specified in number of seconds from Jan. 1 1997. Should you find it onerous to calculate that number yourself, use the sum of the Date() and Time() functions described elsewhere in this document, to set the value for you.

### **Return Type**

None of these functions return a value.

### **Examples**

<b>Event</b>	<b>Enable</b>	<b>Action</b>
F1 Pressed	Default	TrendDefer( 1, Date(2010,1,1) + Time( 11,59,14) )

The above action will start the trending of Channel 1 at 11:59:14 on the first of Jan, 2010.

## TriggerAlarm(*Index*)

Argument	Type	Description
Index	16-bit Unsigned	The index of the alarm to trigger.

### Overview

This function triggers the alarm indicated by the *Index* argument.

### Description

The *Index* parameter should be a number between 1 and 500, and should refer to an alarm within the database's Alarm Table. The alarm will be triggered as if the usual trigger conditions have been met. The alarm will not normally have a controlling expression defined, but this is not a requirement. If the alarm is activated as a result of this function, an "Alarm Activated" event will be posted.

### Return Type

This function does not return a value.

### Examples

Event	Enable	Action
Comms Update	Reg1>100 && Reg2<10	TriggerAlarm(10)

This will set Alarm 10 when the variable Reg1 and Reg2 are both in the indicated states after a communications scan.

## UseAutoEnables(*Port*)

Argument	Type	Description
Port	16-bit Unsigned	The required serial port.

### Overview

This function places the indicated port in "auto-enables" mode.

### Description

This function should only be called for a port to which the Roll-Your-Own Protocol driver has been bound. It has the effect of modifying the way in which the RTS and CTS control lines for the port operate, such that CTS acts as a transmit enable signal, and RTS is gated such that it cannot return to a de-asserted state until any characters in the transmit buffer have cleared the serial port hardware. For RS-485 ports, the CTS line is strapped active and so does not come into the equation, while the RTS line is used to key the transmitter. If half-duplex mode has been enabled, RTS is also used to enable or disable the receiver so as to prevent the terminal from "hearing" its own transmission.

### Return Type

This function does not return a value.

### Examples

#### GLOBAL EVENTS

Event	Enable	Action
System Initialized	Default	UseAutoEnables(2)

This will allow the RTS and CTS on Port 2 to do hardware handshaking.

## UseHalfDuplex(*Port*)

Argument	Type	Description
Port	16-bit Unsigned	The required serial port.

### Overview

This function places the indicated port in half-duplex mode.

### Description

This function should only be called for a port to which the Roll-Your-Own Protocol driver has been bound. It is used to place an RS-485 port into the mode required for two-wire communication, whereby the port's receiver is automatically disabled whenever its transmitter is active. It is nearly always used with "auto-enables" mode, to ensure that the RTS control signal is properly timed with respect to the data stream.

### Return Type

This function does not return a value.

### Examples

#### GLOBAL EVENTS

Event	Enable	Action
System Initialized	Default	UseHalfDuplex(3)

Automatically controls the RTS to indicate to the receiving device that the transmission is complete.

## Val(*Text*)

Argument	Type	Description
Text	String	The string to be converted.

### Overview

**This** function converts a decimal string into a numeric long.

### Description

This function converts its argument into a numeric value, taking the string to be a sequence of decimal digits. It will stop processing the string at the first digit, which cannot be considered valid. Leading spaces are ignored, and leading sign characters are processed providing there are no spaces between the sign and the first digit.

### Return Type

This function returns a 32-bit signed value.

### Examples

Event	Enable	Action
F1 Pressed	Default	A[0] := Val("1234")
F2 Pressed	Default	A[0] := Val("12A4")

In the first case A[0] will equal decimal 1234.

In the second case A[0] will equal decimal 12.

## WriteBlock(*Block*)

Argument	Type	Description
Block	Block Name	The Comms block to be written.

### Overview

This function indicates that the indicated block should be written to the PLC.

### Description

This function instructs the Comms task to write the indicated communications block to the associated device on the next Comms scan. The function returns without waiting for the Comms update to be completed, so you should not assume that the data will be in the PLC as soon as control returns. Neither should you assume that calling the function for two blocks will write the data in the order the functions were called. The WriteBlock function can be called for blocks of any access type, overriding whatever settings are made in the Comms Blocks table.

### Return Type

This function does not return a value.

### Examples

Event	Enable	Action
Page Removed	Default	WriteBlock(Q)

This will write Comms Block Q when a new page has been selected, even if Block Q is selected as a READ Block.

## The Compound Statement

A compound statement is used to include several other statements where only a single statement is otherwise permitted. As an example, the “if” statement controls the execution of a single statement, and yet you may wish to make several actions dependent on the outcome of the conditional expression. By using a compound statement, you can achieve this result by telling EDICT-97 to treat all the actions as one statement.

A compound statement takes the form of an opening curly bracket, followed by any number of other statements. The statement is terminated by a closing curly bracket. As is usual, spaces and carriage returns are not taken into account when parsing the code, but it is conventional to include each element of a compound statement on its own line. The placement of the brackets varies according to programming style, but you will often see indentation used to make it clear which line fall within the compound statement.

### Example 1

```
if( A[0] > 10 ) {  
    TriggerAlarm(1);  
    GotoPage(Page1);  
    Level := A[0];  
}
```

This example shows how curly brackets can be used to make the execution of three action statements conditional upon the expression within the “if” statement. If the brackets had been omitted, the first action alone would have been conditional, and the other two actions would have been executed in all circumstances.

## The If-Else Statement

The if-else statement is used within a program to make the execution of an action conditional upon some expression being *true*. You can optionally add an “else” clause to specify an action to be executed if the expression is *false*. By using compound statement, you can control more than one action at a time.

The controlling expression of an “if” statement can vary in complexity. It can be something as simple as testing a single bit within the PLC, or it can be something much more complex. For example, it is common to see a number of conditions combined with the logical and OR operators.

### Example 1

```
if( A[0] > 10 )  
    Beep(48, 100);  
GotoPage(Page1);
```

This example shows how an “if” statement can be used to control a single action, in this case a function which turns on the terminal’s beeper. Because the “if” controls only a single statement, the GotoPage function is always executed, no matter what value is found in the register. Note how indentation and carriage returns have been used to make this clear to the casual reader, but note also that EDICT-97 takes no notice of such formatting when it compiles your code.

### Example 2

```
if( A[0] > 10 )  
    Beep(48, 100);  
else  
    Beep(60, 100);  
GotoPage(Page1);
```

This example shows a similar construction to the code above, but this time an “else” clause has been added to cause a different pitch of beep to be sounded if the conditional expression turns out to be *false*. As the “else” clause is controlling a single action, the GotoPage function will always be executed.

### Example 3

```
if( A[0] > 10 )  
    Beep(48, 100);  
else {  
    Beep(60, 100);  
    GotoPage(Page1);  
}
```

In this example, curly brackets have been used to group together the second Beep function and the call to GotoPage. As a result of this, they are both considered part of the “else” clause, and so will only be executed if the conditional expression is *false*. This technique uses what is known as a compound statement, whereby a number of actions can be grouped together, and controlled as a single object.

### Example 4

```
if( A[0] > 10 ) {  
    if( B[0] > 10 )  
        Beep(48, 100);  
    else  
        GotoPage(Page2);  
}  
else {  
    Beep(60, 100);  
    GotoPage(Page1);  
}
```

This time, we have used a compound statement for the main “if” section as well, and we have placed a further “if” statement within that statement. This is a technique known as “nesting”, and it allows you to use conditions within conditions. You should always use curly brackets when nesting statements like this, as it makes it clear to the compiler what your intentions are. You may also choose to indent your code to make it easier to read.

## The Loop Statements

Loop statements can be used to execute a given statement or statements repeatedly, until some condition ceases to be *true*. EDICT-97 provides three different loop statements, one of which is really just an abbreviated form of another. Follow the links below to read about each of the loop statements, their syntax and possible applications.

[The While Loop](#)

[The Do-While Loop](#)

[The For Loop](#)

Within the bodies of any of these loop statements, you can make use of the “break” and “continue” statements to modify the loop’s behavior. Executing the “break” statement will cause EDICT-97 to exit from the loop at that point, no matter what the value of the controlling expression. The “continue” statement will cause EDICT-97 to abort this iteration of the loop, and return to the controlling expression immediately, skipping any later statements in the loop body. When either statement is used, it is almost always qualified with an “if” statement.

## The While Loop

The “while” loop is the simplest form of loop statement. It repeatedly tests the value of a controlling expression, and then executes the following statement while ever the expression is *true*. If you want to extend the loop to control more than one statement, enclose the statements in curly brackets to create a compound statement.

### Example 1

```
A[0] := 0;
while( A[0] < 10 ) {
    PrintReport(0, Report1);
    A[0]++;
}
```

In this example, the first line of code loads a value of zero into a register, which is then used to control the loop. While ever this value is less than ten, EDICT-97 will print-out the report as requested, and then add one to the register. The controlling condition will then be tested again, and the loop executed while it remains *true*. In this example, the loop will execute ten times, and so ten copies of the report will be printed. Note that it is possible that the loop body will never execute, as the condition is tested before the loop is run.

## The Do-While Loop

The “do-while” loop is a modified version of the “while” loop whereby the condition is tested at the end of the loop. This implies that the code within the loop will always execute at least once, no matter what the value of the controlling expression. It is used a lot less commonly than the simple “while” loop, but is useful in some situations.

### Example 1

```
do {
    PrintReport(0, Report1);
    Copies++;
} while( Copies < 5 );
```

In this example, the report is printed-out at least once, and then while the value in “Copies” is less than five. Curly brackets have been used to allow two statements to be controlled by the loop, the second of which increments “Copies” on each iteration. This logic implies that the code will print five reports the first time it is called, and then one report each time thereafter.

## The For Loop

If you look again at the example given above for the “while” loop, you will notice that there are three actions performed upon A[0]. It is first loaded with an initial value, it is then used in the controlling expression, and it is finally incremented each time EDICT-97 goes around the loop. As loops of a similar form are so common, EDICT-97 provides a special construction, which allows more compact coding to be used.

Unlike the other loop statements, the “for” statement takes three expressions within round brackets, with a semicolon being used to separate them from each other. The first expression is executed when the loop is initialized, the second is used to control the execution of the loop body just as for a “while” loop, and the third is executed after the loop body on each iteration.

### Example 1

```
for( A[0] := 0; A[0] < 10; A[0]++ )
    PrintReport(0, Report1);
```

This example performs exactly the same operation as the “while” loop example, but the compressed syntax of the “for” loop lets you put all three expressions containing A[0] in the “for” statement itself. As we are only executing a single statement within the loop, no curly brackets are needed in this example. If we had chose, to include the brackets, however, the code would still have worked as intended.

## The Switch Statement

The Switch Statement can be used in the place of a sequence of "if, else if, else if, else" code. Refer to Section B for more information on this Statement.

For more information on "C", you need to obtain a book.

## The Return Statement

The "return" statement is used to abort the current program, and optionally to return a value to the caller. The value will only be accessible to the caller if they used one of the "Call" functions to invoke the program, and will be discarded if the "Run" function is used. The data type of the return value will automatically be converted to the type required by the caller, even if this means a loss of data or accuracy.

### Example 1

```
if( A[0] < 10 )
```

```
    return;
```

In this example, the program execution stops if the value in A[0] is less than 10. Any further statements in the program will be skipped. It is always possible to achieve the same result by placing the other statements in an "if" statement, but using "return" can sometimes make the program easier to read and is often quicker to execute.

### Example 2

```
if( A[0] < 10000 )
```

```
    return Format("0000", A[0]);
```

```
else
```

```
    return "High";
```

This example checks if the value in A[0] is less than 10000, and either returns the value formatted as a four-character string, or the word "High" as appropriate. This program could be called using the CallString function, and the result use in a General Text animation field to provide a form of custom animation.

# Section B - User Programs/Operators/System Variables

## Part 1:Using Programs

### Simple Programs

The simplest form of program comprises a number of actions, with each action being followed by a semicolon, and typically a carriage return. When EDICT-97 executes the program, it performs each of the actions in turn and then returns to the caller. As an example, the program below changes the currently displayed page, and modifies the values held in a pair of Comms block registers...

```
GotoPage(NewPage);
```

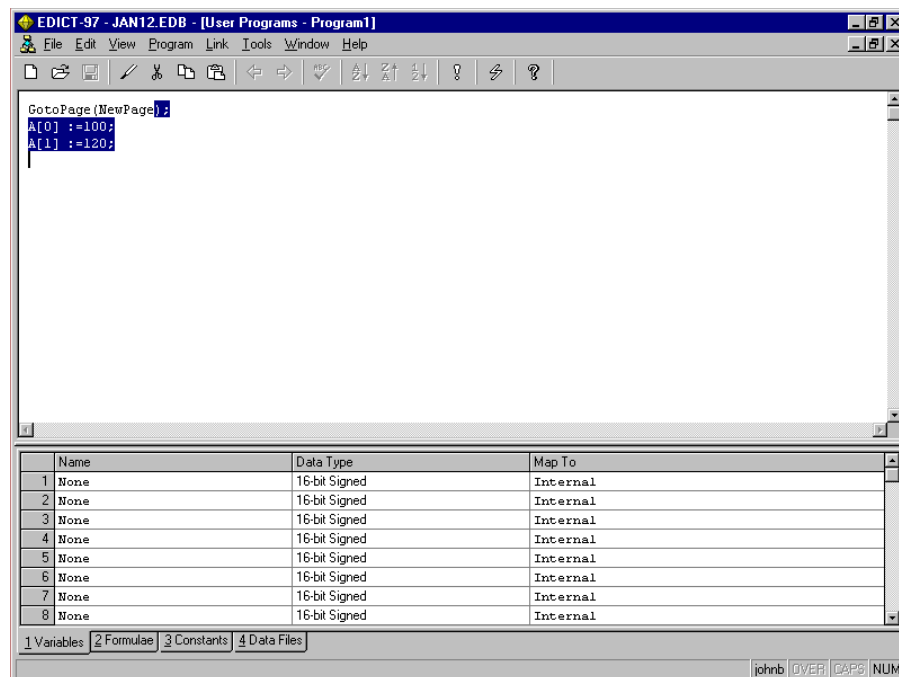
```
A[0] := 100;
```

```
A[1] := 120;
```

### **Complex Programs**

Much more complex programs can be created using a variety of different “statement” types to make decisions, or perform looping operations.

To run programs **Choose User Programs** from the EDICT-97 Menu and type in your program. The following is the simple example listed above.



You must translate your program before you download it to your HMI. From the **User Programs** Window in EDICT-97, **choose Program** then **choose Translate**. After this hit **F9** or the download Icon from your toolbar.

## **Writing Programs**

To provide the ultimate in flexibility, EDICT-97 provides a programming language, similar to the "C" and "Java" languages used in so many applications. Each program is equivalent to a single function within these languages, and may perform a list of actions, controlled by a number of decision-making or loop constructions. Although this manual will not attempt to teach you the subtleties of programming, you can find basic information about programming techniques by reading about the following functions in Section A.

## **Functions**

Run, CallFloat, CallInt, CallLong, CallString, Dispatch

## **Statement Types**

A Program is made up of a number of statements, typically with one statement on each line. The exact formatting in terms of line break and spacing does not actually matter to the compiler, but splitting a program into lines makes it much easier to read. The different Statement types available are: The Action Statement, The Compound Statement, The If-Else Statement, The Loop Statement, The Switch Statement and The Return Statement.

## **The Action Statement**

An action statement is a statement within a program that performs a single action, such as assigning a value to a data item or calling one of EDICT-97's functions. An action statement is the basic type of statement, which makes up a program, as opposed to the various more complex statements, which control execution flow. An action statement takes the form of the action to be completed, followed by a semicolon to terminate the line. The action itself is formatted just as for an action that you might enter into one of EDICT-97's event maps.

### **Example 1**

```
A[0] := 10;
B[0] := 12 * B[7] + B[3];
PrintReport(Report1);
GotoPage(Page1);
```

## **The Compound Statement**

A compound statement is used to include several other statements where only a single statement is otherwise permitted. As an example, the "if" statement controls the execution of a single statement, and yet you may wish to make several actions dependent on the outcome of the conditional expression. By using a compound statement, you can achieve this result by telling EDICT-97 to treat all the actions as one statement.

A compound statement takes the form of an opening curly bracket, followed by any number of other statements. A closing curly bracket terminates the statement. As is usual, spaces and carriage returns are not taken into account when parsing the code, but it is conventional to include each element of a compound statement on its own line. The placement of the brackets varies according to programming style, but you will often see indentation used to make it clear, which lines fall within the compound statement.

### **Example 1**

```
if( A[0] > 10 ) {
    TriggerAlarm(1);
    GotoPage(Page1);
    Level := A[0];
}
```

This example shows how curly brackets can be used to make the execution of three action statements conditional upon the expression within the “if” statement. If the brackets had been omitted, the first action alone would have been conditional, and the other two actions would have been executed in all circumstances.

### **The If-Else Statement**

The if-else statement is used within a program to make the execution of an action conditional upon some expression being *true*. You can optionally add an “else” clause to specify an action to be executed if the expression is *false*. By using compound statements, you can control more than one action at a time.

The controlling expression of an “if” statement can vary in complexity. It can be something as simple as testing a single bit within the PLC, or it can be something much more complex. For example, it is common to see as a number of conditions combined with the logical AND and OR operators.

#### **Example 1**

```
if( A[0] > 10 )  
    Beep(48, 100);  
GotoPage(Page1);
```

This example shows how an “if” statement can be used to control a single action, in this case a function which turns on the terminal’s beeper. Because the “if” controls only a single statement, the GotoPage function is always executed, no matter what value is found in the register. Note how indentation and carriage returns have been used to make this clear to the casual reader, but note also that EDICT-97 takes no notice of such formatting when it compiles your code.

#### **Example 2**

```
if( A[0] > 10 )  
    Beep(48, 100);  
else  
    Beep(60, 100);  
GotoPage(Page1);
```

This example shows a similar construction to the code above, but this time an “else” clause has been added to cause a different pitch of beep to be sounded if the conditional expression turns out to be *false*. As the “else” clause is controlling a single action, the GotoPage function will always be executed.

#### **Example 3**

```
if( A[0] > 10 )  
    Beep(48, 100);  
else {  
    Beep(60, 100);  
    GotoPage(Page1);  
}
```

In this example, curly brackets have been used to group together the second Beep function and the call to GotoPage. As a result of this, they are both considered part of the “else” clause, and so will only be executed if the conditional expression is *false*. This technique uses what is known as a compound statement, whereby a number of actions can be grouped together, and controlled as a single object.

#### Example 4

```
if( A[0] > 10 ) {  
    if( B[0] > 10 )  
        Beep(48, 100);  
    else  
        GotoPage(Page2);  
}  
else {  
    Beep(60, 100);  
    GotoPage(Page1);  
}
```

This time, we have used a compound statement for the main “if” section as well, and we have placed a further “if” statement within that statement. This is a technique known as “nesting”, and it allows you to use conditions within conditions. You should always use curly brackets when nesting statements like this, as it makes it clear to the compiler what your intentions are. You may also choose to indent your code to make it easier to read.

### **The Loop Statements**

Loop statements can be used to execute a given statement or statements repeatedly, until some condition ceases to be *true*. EDICT-97 provides three different loop statements, one of which is really just an abbreviate form of another. Follow the links below to read about each of the loop statements, their syntax and possible applications.

[The While Loop](#)

[The Do-While Loop](#)

[The For Loop](#)

Within the bodies of any of these loop statements, you can make use of the “break” and “continue” statements to modify the loop’s behavior. Executing the “break” statement will cause EDICT-97 to exit from the loop at that point, no matter what the value of the controlling expression. The “continue” statement will cause EDICT-97 to abort this iteration of the loop, and return to the controlling expression immediately, skipping any later statements in the loop body. When either statement is used, it is almost always qualified with an “if” statement.

### **The While Loop**

The “while” loop is the simplest form of loop statement. It repeatedly tests the value of a controlling expression, and then executes the following statement while ever the expression is *true*. If you want to extend the loop to control more than one statement, enclose the statements in curly brackets to create a compound statement.

#### Example 1

```
A[0] := 0;  
while( A[0] < 10 ) {  
    PrintReport(0, Report1);  
    A[0]++;  
}
```

In this example, the first line of code loads a value of zero into a register, which is then used to control the loop. While ever this value is less than ten, EDICT-97 will print out the report as requested, and then add one to the register. The controlling condition will then be tested again, and the loop executed while it remains *true*. In this example, the loop will execute ten times, and so ten copies of the report will be printed. Note that it is possible that the loop body will never execute, as the condition is tested before the loop is run.

### **The Do-While Loop**

The “do-while” loop is a modified version of the “while” loop whereby the condition is tested at the end of the loop. This implies that the code within the loop will always execute at least once, no matter what the value of the controlling expression. It is used a lot less commonly than the simple “while” loop, but is useful in some situations.

#### **Example 1**

```
do {  
    PrintReport(0, Report1);  
    Copies++;  
} while( Copies < 5 );
```

In this example, the report is printed out at least once and then while ever the value in “Copies” is less than five. Curly brackets have been used to allow two statements to be controlled by the loop, the second of which increments “Copies” on each iteration. This logic implies that the code will print five reports the first time it is called and then one report each time thereafter.

### **The For Loop**

If you look again at the example given above for the “while” loop, you will notice that there are three actions performed upon A[0]. It is first loaded with an initial value, it is then used in the controlling expression, and it is finally incremented each time EDICT-97 goes around the loop. As loop of a similar form are so common, EDICT-97 provides a special construction, which allows more compact coding to be used.

Unlike the other loop statements, the “for” statement takes three expressions within round brackets, with a semicolon being used to separate them from each other. The first expression is executed when the loop is initialized, the second is used to control the execution of the loop body just as for a “while” loop, and the third is executed after the loop body on each iteration.

#### **Example 1**

```
for( A[0] := 0; A[0] < 10; A[0]++ )  
    PrintReport(0, Report1);
```

This example performs exactly the same operation as the “while” loop example, but the compressed syntax of the “for” loop lets you put all three expressions containing A[0] in the “for” statement itself. As we are only executing a single statement within the loop, no curly brackets are needed in this example. If we had chose, to include the brackets, however, the code would still have worked as intended.

### **The Switch Statement**

The Switch Statement can be used in the place of a sequence of “if, else if, else if, else” code. The Switch Statement may take this basic form:

```
Switch( value ) {  
    case 1:  
        program text for value = 1;  
        more program text for value = 1;  
        break;  
    case 7:  
        program text for value = 7;  
        break;  
    case 3:  
    case 4:  
        program text for value = 3 or value = 4;
```

```

        break;
default:
    program text for what to do if value = none of the above;
    break;
}

```

Forgetting to put in the break statements will let succeeding code be executed.  
For more information on “C”, you need to obtain a book.

### **The Return Statement**

The “return” statement is used to abort the current program, and optionally to return a value to the caller. The value will only be accessible to the caller if they used one of the “Call” functions to invoke the program, and will be discarded if the “Run” function is used. The data type of the return value will automatically be converted to the type required by the caller, even if this means a loss of data or accuracy.

#### **Example 1**

```

if( A[0] < 10 )
    return;

```

In this example, the program execution stops if the value in A[0] is less than 10. Any further statements in the program will be skipped. It is always possible to achieve the same result by placing the other statements in an “if” statement, but using “return” can sometimes make the program easier to read and is often quicker to execute.

#### **Example 2**

```

if( A[0] < 10000 )
    return Format("0000", A[0]);
else
    return "High";

```

This example checks if the value in A[0] is less than 10000, and either returns the value formatted as a four-character string, or the word “High” as appropriate. This program could be called using the CallString function, and the result used in a General Text animation field to provide a form of custom animation.

### **Using Actions**

Actions are the means by which you instruct EDICT-97 to do something. An action can either be a section of code, which modifies a data value, or a call to an active function. Active functions are those functions which themselves change data values, or cause a change of state within EDICT-97. Follow each of the links below for more details of each type of action...

[1\)Modifying Data](#)

[2\)Modifying Bits](#)

[3\)Using Functions](#)

#### **1) Modifying Data**

The most common way of modifying a data value is to use code similar to...

```
A[0] := 100 * B[0]
```

The left-hand side of the “:=” assignment operator can be replaced with any expression which refers to a writable value, and the right-hand side of the operator can be replaced with any expression which produces a result of a suitable data type.

There are a number of more specialized assignment operators, which can take the value in a location and combine it with another expression using one of a number of operators, before storing the value back in the original location. These exist simply to save on typing, and the same effect can always be achieved using a simple assignment.

Finally, EDICT-97 provides special operators to allow you to increase or decrease a data value by a value of one. These operators, known as the increment and decrement operators respectively, provide a useful shorthand method of achieving this common end. The examples below show how to increment or decrement A[0]...

```
A[0]++
```

```
A[0]--
```

## **2) Modifying Bits**

If you want to modify a single bit within a data value, you can use the bit selection operator “.” to indicate which bit you wish to modify. As an example, the examples below turn the second bit of register A[0] on and off respectively...

```
A[0].1 := 1
```

```
A[0].1 := 0
```

The expression to the left-hand side of the bit selection operator can be any modifiable integer value, while the expression to the right-hand side can be either a constant value or another integer expression. This last technique can be used to use one expression to select a bit within another, a process known as “bit indirection”.

## **3) Using Functions (See Section A for list of Functions)**

EDICT-97 provides a number of functions, which you can call from within your programs, expressions and actions. A function is invoked by following its name with an opening round bracket, listing any arguments with comma as separators, and following the sequence with a closing round bracket. If the function takes no arguments, the opening and closing brackets must still follow it. EDICT-97 supports a concept called “function overloading”, where a given function may be able to take arguments of different types, and may even support optional arguments. Functions are divided into two classes; namely, active and passive functions. An active function either changes data, or causes a change of state within EDICT-97. It can be used to form an action or to form an action statement within a program. Active functions cannot be used in expressions, where changing data is not permitted. Passive functions do not change anything within EDICT-97, but simply return a value based upon their arguments. For example, Min is a function, which returns the lower of its two arguments. Passive functions are allowed within expressions, but may not, on their own, form actions.

### **Examples**

```
GotoPage(Page1)      // Function invocation in an action
```

```
StopSystem();        // Function invocation as program statement
```

```
Min(A[1], A[2])       // Function invocation in an expression
```

## **Using Expressions**

Expressions are used throughout EDICT-97 whenever a data value is required. An expression is a combination of data items, known as operands, with special symbols known as operators. These symbols are used to represent common operations such as addition and subtraction, as well as more complex operations like bit shifting and so on. The simplest form of expression can be a single data item, while more complex expression may contain many data items, combined with a large number of operators.

## Examples

Code	Description
100	A constant integer value of 100.
A[0]	The first element in Comms block A.
[D100]	Register D100 in the first Comms device.
A[0] * 12 / 5	Register A[0] with a scale-factor.
C[5].6	Bit 6 of the indicated Comms block item.
B[A[0]]	The A[0] <sup>th</sup> element in Comms block B.

The final example shows how to use an expression to select an item from within a communications block, using a process known as "indirect addressing". For more information on this powerful technique, follow the link below to view a list of possible operands, and then look under Comms Block Data.

## Comments

EDICT-97 allows you to include comments in your code. You can include a so-called block comment between the character sequences `/*` and `*/`. Such a comment is able to include line breaks and so can span multiple lines within a program. You can also introduce a single-line comment using the `//` character sequence, and then terminate the comment with a line break. This construction lets you add a comment to the end of a line of program code, or to the end of an expression or action.

## Examples

```
A[0] := 10; /* This is a comment */  
A[1] := 20; // This is a comment too
```

## Type Names

You have selected a keyword, which is a data type name. Since EDICT-97 does not allow local variables to be declared within a program body, these keywords are used only in type conversion sequences, known as type casts.

## Data Types

The compiler used within EDICT-97 supports a number of data types, including a wide variety of integer types, a floating point type and a dynamic string type. The floating-point type uses 32-bit IEEE representation to hold values with an accuracy of around 7 significant figures. Stored string variables can be up to 256 characters in length, although intermediate values may exceed this length considerably. The table below lists the integer data types, together with the range of value each can hold.

Type	Range
16-bit Unsigned Value	0 to 65535
32-bit Unsigned Value	0 to 4294967295
16-bit Signed Value	-32768 to +32767
32-bit Signed Value	-2147483648 to +2147483647

In general, EDICT-97 will look after conversions between types as and when required by the context, and will automatically "promote" a data value to the next larger type should the current type prove too small to hold an intermediate value during a calculation. If you need to perform an explicit type conversion, you can use what is known as a type cast sequence.

### **Type Casting**

If you need explicitly to change the data type of an expression, you can use what is known as a type cast sequence. This sequence can take two forms, one being the traditional “C” format and the other being that supported by “C++” and Java. The first syntax is formed from a type name contained within a pair of round brackets, and is shown in the first pair of examples. The second form uses the type name in the same way you would invoke a function, and is shown in other examples.

The type names, which can be used in type sequences, are...

<b>Keyword</b>	<b>Resulting Type</b>
Uint	16-bit Unsigned Value
Ulong	32-bit Unsigned Value
int	16-bit Signed Value
long	32-bit Signed Value
float	Floating Point Value

Examples  $B[0] := C[0] * (\text{long}) A[0]$   $D[0] := E[0] + (\text{uint}) (A[0] / 2.5)$   $B[0] := C[0] * \text{long}(A[0])$   $D[0] := E[0] + \text{uint}(A[0] / 2.5)$

### **The “break” Keyword**

The “break” keyword has two distinct purposes. The first is to cause a premature exit from the current loop construction, irrespective of the value of the expression controlling the loop. As an example, you may use this keyword to stop a search early should you find some particular data value in an array. The second purpose is to separate the “case” clauses of a “switch” statement, and to prevent the flow of execution continuing from one clause to the next. For examples of both uses, please review **The Loop Statements** and **The Switch Statement** on pages 4 to 6 of this section.

### **The “case” Keyword**

The “case” keyword is used to introduce a specific value-matching clause within a “switch” statement. A constant expression and a colon must follow it. For an example of how it is used, please review **The Switch Statement** on page 5 of this section.

### **The “continue” Keyword**

The “continue” keyword is used within a loop construct to indicate that execution flow should return to the top of the loop, and that any further statements within the loop body should be skipped. For an example of how it is used, please review **The Loop Statements** on pages 4 and 5 of this section.

### **The “default” Keyword**

The “default” keyword is used to introduce a default action clause within a “switch” statement. Unlike “C” or Java, EDICT-97 insists that a “default” statement form the last clause of the statement. For an example of how it is used, please review **The Switch Statement** on page 5 of this section.

### **The “do” Keyword**

The “do” keyword is used to introduce a loop construct, whereby a section of code can be repeated a number of times, based upon the value of a controlling expression. The controlling expression is evaluated at the end of the loop, so the code must execute at least once. For an example of how it is used, please review **The Loop Statements** on pages 4 and 5 of this section.

### **The “else” Keyword**

The “else” keyword is used to introduce an else-clause for an “if” statement. This allows a section of code to be executed if the conditional expression within the “if” statement does not evaluate to *true*. For an example of how it is used, please review **The If-Else Statement** on page 3 of this section.

### **The “for” Keyword**

The “for” keyword is used to form a loop construct, wherein it is possible to specify initialization, controlling and iteration expressions of a single statement. The controlling expression is evaluated before the loop body is executed, so the code within the body may not be executed at all. For an example of how it is used, please review **The Loop Statements** on pages 4 and 5 of this section.

### **The “if” Keyword**

The “if” keyword is used to introduce a statement whereby a section of code may or may not be executed, based on whether a controlling expression is *true* or *false*. The “else” keyword may also be used to introduce code to be executed should the main body not be executed. For an example of how it is used, please review **The If-Else Statement** on page 3 of this section.

### **The “return” Keyword**

The “return” statement is used to terminate a program early, and optionally to return a value to be passed back to the caller. The value returned may be of any data type, but will be converted to the type specifically requested by the caller. For an example of how it is used, please review **The Return Statement** on page 6 of this section.

### **The “switch” Keyword**

The “switch” keyword is used to introduce a statement whereby an expression can be compared against a number of pre-defined constant values, and different sections of code executed as a result. For an example of how it is used, please review **The Switch Statement** on page 5 of this section.

### **The “while” Keyword**

The “while” keyword has two purposes. First, it can be used to terminate a loop construct introduced with a “do” keyword. Second, it can be used to introduce a loop construct of its own, whereby a section of code can be repeated a number of times, based upon the value of a controlling expression. The controlling expression is evaluated at the start of the loop, so the code may not execute at all. For an example of how it is used, please review **The Loop Statements** on page B4 of this section.

### **The “WAS” Keyword**

The “WAS” keyword serves a special purpose; namely, to mark a section of code as containing errors, and to prevent it from generating further error messages when it is compiled. It can thus be considered as a form of comment marker. The keyword is added by EDICT-97 when it detects that a change you have made somewhere in the database will cause existing code to become invalid. Rather than simply deleting the code, EDICT-97 prefixes it with “WAS” to remove the error while still allowing you to see the previous code. The “Recompile” command on the “Tools” menu will remove “WAS” keywords from the code before recompiling if the previous errors have now been corrected.

## Part 2: Operators

Operators are used to combine data items, or to modify a single data item. The data items manipulated by operators are called operands. An operator who combines two data items is called a binary operator, while those that work on a single data item are called unary operators.

Examples of binary operators are the + operator used to add two data items, and the \* operator used to multiply two values. An example of the unary operator is ~, which is used to take the one's complement of an operand.

Operators are arranged in what are called priority groups. These groups are used to control the order in which operators are applied. For example, it is normal to apply multiplication operators before addition operators, and this is rule enforced by placing the latter in a lower priority group than the former. The table below lists the groups, together with the operators they contain. You can click on a given group to see more details about the operators and their functions.

Group	Operators
<a href="#">Group 1</a>	++ -- . [
<a href="#">Group 2</a>	(type) ++ -- ! ~ - +
<a href="#">Group 3</a>	* / %
<a href="#">Group 4</a>	+ -
<a href="#">Group 5</a>	<< >>
<a href="#">Group 6</a>	< > <= >=
<a href="#">Group 7</a>	== !=
<a href="#">Group 8</a>	&
<a href="#">Group 9</a>	
<a href="#">Group 10</a>	^
<a href="#">Group 11</a>	&&
<a href="#">Group 12</a>	
<a href="#">Group 13</a>	:= op=
<a href="#">Group 14</a>	,

### Operator Group 1

Operator	Type	Name	Used With
++	Postfix	Increment.	Numerics.
--	Postfix	Decrement.	Numerics.
.	Binary	Bit selection.	Integers.
[	Special	Indexing.	See Text.

The **increment** and **decrement** operators used to add or subtract one to the contents of a modified value. As they are postfix operators, they follow the operand, which they are to modify, and their return value is that of the operand before it is modified. You should contrast this behavior with the prefix versions contained in the next group.

The **bit selection** operator is used to examine the value of a bit within an integer value, or to specify a bit within a modifiable value to which a new value is to be assigned. Both operands must be integers. The return value of the operator is either 1 or 0, depending on the value of the bit being examined.

The **indexing** operator is a special operator, applied using syntax of "L[R]", where L and R are the left-hand and right-hand operands respectively. The operator is used to select elements from an array or a string, with the R<sup>th</sup> element being chosen from the item specified by L. Note that a value of zero in R selects the first element.

## Examples

Code	Description
A[0]++	The value in A[0] is increased by one.
A[0]--	The value in A[0] is decreased by one.
A[0].4	Returns the value of bit 4 of A[0].
A[5]	Returns the value of the 6 <sup>th</sup> element of A.
"Fred"[2]	Returns the ASCII value of "e".

## Operator Group 2

Operator	Type	Name	Used With
(type)	Prefix.	Type Cast.	Anything.
++	Prefix.	Increment.	Numerics.
--	Prefix.	Decrement.	Numerics.
!	Prefix.	Logical NOT.	Numerics.
~	Prefix.	Bitwise NOT.	Integers.
-	Prefix.	Unary Minus.	Numerics.
+	Prefix.	Unary Plus.	Numerics.

The **type cast** operator is used to change the data type of its operand. The keyword within the brackets should be one of the type names supported by EDICT-97. For more information on how to use type casts, please follow [this link](#) to see a specific section, where you will find details of the types available and the permitted type conversions.

The **increment** and **decrement** operators used to add or subtract one to the contents of a modified value. As they are prefix operators, they precede the operand, which they are to modify, and their return value is that of the operand after it is modified. You should contrast this behavior with the postfix versions contained in the previous group.

The **logical NOT** operator returns a value of one if its operand is zero, and a value of zero in all other cases. It can be used as a shorthand method of comparing a value with zero, or a way of inverting the logic of an expression. The **bitwise NOT** returns a value equal to the original operand with the state of every data bit inverted.

The **unary minus** operator changes the sign of the operand. It cannot be applied to unsigned constants, and applying it to an unsigned variable will first convert the operand to a signed value. The **unary plus** operator is effectively a null operator, as it simply returns the value of its operand.

## Examples

Code	Description
A[0]++	The value in A[0] is increased by one.
A[0]--	The value in A[0] is decreased by one.
!A[0]	Returns 1 if A[0] is zero, or 0 otherwise.
~0x7777	Returns the one's complement of 0x7777.
-A[0]	Returns the two's complement of A[0].

## Operator Group 3

Operator	Type	Name	Used With
*	Binary.	Multiplication.	Numerics.
/	Binary.	Division.	Numerics.
%	Binary.	Remainder.	Integers.

The **multiplication** operator returns the product of its operands, promoting the result to a larger data type should this be required. The **division** operator returns the quotient of its operands, or the largest possible value for the data type in question if the second operand is zero. The **remainder** operator returns the remainder of dividing its first operand by its second, or zero if the second operand is zero.

#### Examples

Code	Description
A[0] * 10	Returns A[0] multiplied by 10.
1234 / 10	Returns a value of 123.
1234 / 10.0	Returns a value of 123.4.
A[0] % 10	Returns the last decimal digit of A[0].

#### Operator Group 4

Operator	Type	Name	Used With
+	Binary.	Addition.	Numerics, Strings.
-	Binary.	Subtraction.	Numerics.

The **addition** operator returns the sum of its operands, promoting the result to a larger data type should this be required. The **subtraction** operator returns the difference of its operands, again promoting the resulting data type if required.

#### Examples

Code	Description
A[0] + A[1]	Returns the sum of A[0] and A[1].
A[0] - 10	Returns 10 less than the value in A[0].

#### Operator Group 5

Operator	Type	Name	Used With
<<	Binary.	Shift Left.	Integers.
>>	Binary.	Shift Right.	Integers.

The **shift left** operator returns a value equal to its left-hand operand shifted left by the number of bits specified by its right-hand operand. Empty bits are always filled with zeros. This is the equivalent of multiplying the value by a given power of two.

The **shift right** operator returns a value equal to its left-hand operand shifted right by the number of bits specified by its right-hand operand. Empty bits are always filled with zeros. This is the equivalent of dividing the value by a given power of two.

#### Examples

Code	Description
A[0] << 2	Returns A[0] multiplied by 4.
A[0] >> 2	Returns A[0] divided by 4.

### Operator Group 6

Operator	Type	Name	Used With
<	Binary.	Less Than.	Numerics, Strings.
<=	Binary.	Less or Equal.	Numerics, Strings.
>	Binary.	Greater Than.	Numerics, Strings.
>=	Binary.	Greater or Equal.	Numerics, Strings.

The **inequality** operators perform the specified comparison between their operands, and return 1 if the inequality holds, and 0 if it does not. For string values, the calculation is based upon a case-sensitive ASCII comparison, such that "A" is considered less than "B", and "B" is considered less than "BB".

#### Examples

Code	Description
A[0] > 10	Returns 1 if A[0] is greater than 10.
Name < "Z"	Returns 1 if Name is less than "Z".

### Operator Group 7

Operator	Type	Name	Used With
==	Binary.	Equal To.	Numerics, Strings.
!=	Binary.	Not Equal To.	Numerics, Strings.

The **equality** operators perform the specified comparison between their operands, and return 1 if the condition holds, and 0 if it does not. For string values, the calculation is based upon a case-sensitive ASCII comparison, so that strings are considered the same if and only if they contain the exact same sequence of characters.

#### Examples

Code	Description
A[0] == 10	Returns 1 if A[0] is equal to 10.
Name != "Mike"	Returns 1 if Name is not equal to "Mike".

### Operator Group 8

Operator	Type	Name	Used With
&	Binary.	Bitwise AND.	Integers.

The **bitwise AND** operator combines its two operands using the Boolean AND operation, whereby a bit in the result is set if and only if both corresponding bits in the operands are set. This operator is often used to select a number of bits from a value, or to turn a bit off by means of a bit mask. Compare with the logical AND operator in Group 11.

#### Examples

Code	Description
A[0] & 0xFF	Returns the bottom byte of A[0].

### Operator Group 9

Operator	Type	Name	Used With
	Binary.	Bitwise OR.	Integers.

The **bitwise OR** operator combines its two operands using the Boolean OR operation, whereby a bit in the result is set if either or both of the corresponding bits in the operands are set. This operator is often used to combine two bit-mapped values, or to turn a bit on by means of a bit mask. Compare with the logical OR operator in Group 12.

#### Examples

Code	Description
A[0]   0x01	Returns A[0] with bit 0 turned on.

### Operator Group 10

Operator	Type	Name	Used With
^	Binary.	Bitwise XOR.	Integers.

The **bitwise XOR** operator combines its two operands using the Boolean Exclusive OR operation, whereby a bit in the result is set one and one alone of the corresponding bits in the operands are set. This operator is often used to invert the state of a given bit or bits by means of a bit mask containing the bits to be changed.

#### Examples

Code	Description
A[0] ^ 0x10	Returns A[0] with bit 4 inverted.

### Operator Group 11

Operator	Type	Name	Used With
&&	Binary.	Logical AND.	Numerics.

The **logical AND** operator returns a value of one if both of its operands are not equal to zero, or a value of zero in all other cases. Unlike the bitwise AND operator, it does not consider the individual bits of its operands, but simply whether they are *true* or *false*. This is the operator you should use to combine conditions in The If-Else Statement and the like, and in all cases where a bitwise operation is not specifically needed.

**NOTE:** EDICT-97's compiler will always execute both operand of this operator, even if the first one is *false*. This is different from the behavior of "C" or Java compilers, which will stop executing the operands once the result has become clear.

#### Examples

Code	Description
!V1 && V2 > 1	Return 1 if V1 is zero, and V2 is greater than 1.

## Operator Group 12

Operator	Type	Name	Used With
<code>  </code>	Binary.	Logical OR.	Numerics.

The **logical OR** operator returns a value of one if either of its operands are not equal to zero, or a value of zero in all other cases. Unlike the bitwise OR operator, it does not consider the individual bits of its operands, but simply whether they are *true* or *false*. This is the operator you should use to combine conditions in The If-Else Statement and the like, and in all cases where a bitwise operation is not specifically needed.

**NOTE:** EDICT-97's compiler will always execute both operands of this operator, even if the first one is *true*. This is different from the behavior of "C" or Java compilers, which will stop executing the operands once the result has become clear.

### Examples

Code	Description
<code>!V1    V2 &gt; 1</code>	Return 1 if V1 is zero, or V2 is greater than 1.

## Operator Group 13

Operator	Type	Name	Used With
<code>:=</code>	Binary.	Assignment.	Anything.
<code>op=</code>	Binary.	Compound Assign.	Anything.

The **assignment** operator is used to assign a value to a modifiable value. It places the value of its right-hand operand into the location specified by the left-hand operand, and returns a value equal to the new value. Returning a value like this lets you "chain" assignments together, as shown in the second example below. The operator is also special in that it is evaluated from right-to-left, again to allow chaining.

The **compound assignment** operator is really a family of operators. The "*op*" in the syntax shown above should be replaced by any binary operator detailed in the early groups, with the exception of the logical OR and AND operators. The operator takes the value in its two operands and applies the operator in question, before storing the value in the left-hand operand. The code "*A op= B*" is thus equivalent to "*A := A op B*".

### Examples

Code	Description
<code>A[0] := 10</code>	Sets A[0] equal to 10.
<code>V1 := V2 := 0</code>	Sets both the variables V1 and V2 to zero.
<code>B[0].2 := 1</code>	Turns on bit 2 in register B[0].
<code>B[0].2 := 0</code>	Turns off bit 2 in register B[0].
<code>C[0] *= 10</code>	Sets C[0] equal to itself multiplied by 10.
<code>C[0] += 40</code>	Sets C[0] equal to itself plus 40.
<code>C[0] ^= 0x04</code>	Flips the value of bit 2 of C[0].

### **Operator Group 14**

<b>Operator</b>	<b>Type</b>	<b>Name</b>	<b>Used With</b>
,	Binary.	Sequence.	Anything.

The **sequence** operator returns the value of its second operand, after evaluating the operands in order. The operator is not used within “normal” expressions, but may be used to include two separate sections of code where only a single expression is expected, such as in expressions forming a “for” statement.

### **Examples**

<b>Code</b>	<b>Description</b>
A[0]++, B[0]++	Increments A[0] and B[0], returning B[0].

## **Part 3: System Variables**

EDICT-97 provides a number of system variables, which are used either to reflect the state of the system, or to modify the behavior of the system in some way. The former type of variable will be read-only, while the latter type can have a value assigned to it. Follow the link below to get a full list of system variables, with examples as to their use.

### **System Variable Index**

The table below lists the available system variables...

<b>Variable Name</b>	<b>Page</b>
<u>ActiveAlarms</u>	B17
<u>CommsError</u>	B18
<u>CommsUpdate</u>	B18
<u>Date</u>	B18
<u>Day</u>	B19
<u>DispBrightness</u>	B19
<u>DispContrast</u>	B19
<u>DisplInvert</u>	B19
<u>FALSE</u>	B20
<u>Hours</u>	B20
<u>IsBatteryLow</u>	B20
<u>IsSirenOn</u>	B20
<u>Mins</u>	B20
<u>Month</u>	B21
<u>Secs</u>	B21
<u>TRUE</u>	B21
<u>UnacceptedAlarms</u>	B22
<u>Year</u>	B22

### **ActiveAlarms**

**Data Type**

16-bit Unsigned.

**Write Status**

read-only.

**Description**

The number of alarms currently active within the system.

### **CommsError**

**Data Type**

32-bit Unsigned.

**Write Status**

read-only.

**Description**

Bit 0 of this variable will be set if any Comms errors are present. Bits 1 through 20 will be set if the corresponding device in the device table is not responding to any Comms requests, and is thus assumed to have been disconnected. Bit 30 will be set if there are any Comms errors associated with the user-defined block table, while bit 31 will be set if there are Comms errors associated with the automatic block table.

### **CommsUpdate**

**Data Type**

16-bit Unsigned.

**Write Status**

read-only.

**Description**

The current Comms update time in milliseconds.

### **Date**

**Data Type**

16-bit Unsigned.

**Write Status**

read-only.

**Description**

The day-of-the-month element of the current time and date.

## **Day**

### **Data Type**

16-bit Unsigned.

### **Write Status**

read-only.

### **Description**

The day-of-the-week element of the current time and date.

## **DispBrightness**

### **Data Type**

16-bit Unsigned.

### **Write Status**

Read-Write.

### **Description**

The brightness of the terminal's display, in percentage terms. Not all terminals are capable of controlling their display brightness, and some may only support on and off operation. The terminal will generally honor the setting to the best of its ability.

## **DispContrast**

### **Data Type**

16-bit Unsigned.

### **Write Status**

Read-Write.

### **Description**

The contrast of the terminal's display, in percentage terms. 50% is the default setting, to be used in "standard" viewing conditions. Not all terminals are capable of controlling their display contrast. The terminal will generally honor the setting to the best of its ability.

## **DisplInvert**

### **Data Type**

16-bit Unsigned.

### **Write Status**

Read-Write.

### **Description**

If this variable is set to a non-zero value, the terminal's display will be inverted. Not all terminals are capable of supporting this feature. The terminal will generally honor the setting to the best of its ability.

## **FALSE**

### **Data Type**

16-bit Unsigned.

### **Write Status**

read-only.

### **Description**

A constant value of 0.

## **Hours**

### **Data Type**

16-bit Unsigned.

### **Write Status**

read-only.

### **Description**

The hours element of the current time and date.

## **IsBatteryLow**

### **Data Type**

16-bit Unsigned.

### **Write Status**

read-only.

### **Description**

A non-zero value if the terminal's battery is low, or zero otherwise.

## **IsSirenOn**

### **Data Type**

16-bit Unsigned.

### **Write Status**

read-only.

### **Description**

A non-zero value if the internal sounder is activated, or zero otherwise.

## **Mins**

### **Data Type**

16-bit Unsigned.

### **Write Status**

read-only.

### **Description**

The minutes element of the current time and date.

### **Month**

**Data Type**

16-bit Unsigned.

**Write Status**

read-only.

**Description**

The month element of the current time and date.

### **Param**

**Data Type**

32-bit Unsigned.

**Write Status**

read-only.

**Description**

A variable that temporarily holds a value representing the key pressed, or touch-screen location.

### **pi**

**Data Type**

Float

**Write Status**

read-only.

**Description**

The value of PI ( 3.14159....)

### **Secs**

**Data Type**

16-bit Unsigned.

**Write Status**

read-only.

**Description**

The seconds element of the current time and date.

### **TRUE**

**Data Type**

16-bit Unsigned.

**Write Status**

read-only.

**Description**

A constant value of 1.

### **UnacceptedAlarms**

**Data Type**

16-bit Unsigned.

**Write Status**

read-only.

**Description**

The number of active alarms which have not been accepted.

### **Year**

**Data Type**

16-bit Unsigned.

**Write Status**

read-only.

**Description**

The 2 digit year element of the current time and date.

### **Year4**

**Data Type**

16-bit Unsigned.

**Write Status**

read-only.

**Description**

The 4 digit year element of the current time and date.

# Section C - Error Codes

## Lexical Error Index

This table contains all the error messages, which can be generated by the low-level section of the compiler known as the “lexical analyzer”.

<u>Character Constant Too Long</u>	<u>Empty Character Constant</u>	<u>Identifier Too Long</u>
<u>Invalid Decimal Point</u>	<u>Invalid Digit in Constant</u>	<u>Invalid Digit in Escape Sequence</u>
<u>Invalid Escape Sequence</u>	<u>Invalid Operator</u>	<u>Invalid Suffix on Constant</u>
<u>New Line in Character Constant</u>	<u>New Line in Escape Sequence</u>	<u>New Line in String Constant</u>
<u>No Digits After Decimal Point</u>	<u>Open Block Comment</u>	<u>Overflow in Escape Sequence</u>
<u>Repeated Suffix on Constant</u>	<u>String Constant Too Long</u>	<u>Unexpected Character</u>

### Internal Error

An unexpected error has occurred within the lexical analyzer, the component of the compiler which breaks-up the incoming stream into “tokens” to be processed by the rest of the compiler. This error may be caused by a lack of memory, or by incorrect installation of the software. It may also be caused by a bug within the software, and we would thus ask you to send a copy of the file, which caused the problem to our Technical Support department for study.

### Invalid Decimal Point

You have entered a decimal point within a numeric constant, which has a prefix indicating that it is not a decimal number. Decimal points are not allowed within hexadecimal, octal or binary constants.

### Invalid Digit in Constant

You have entered a numeric constant, which contains a character, which is not valid when considered along with any radix specifier you may have used. For example, you may not use the characters “A” to “F” in anything other than hex constants, or the digits “8” or “9” within octal constants.

### **Repeated Suffix on Constant**

You have used the same suffix twice on a numeric constant. The “U” suffix is used to specify an unsigned number or the “L” suffix is used to specify a 32-bit number, and each should be used at most once on a given constant.

### **Invalid Suffix on Constant**

You have used an invalid suffix letter on a numeric constant. The only valid suffixes are the “U” suffix used to specify an unsigned number and the “L” suffix used to specify a 32-bit number. It is possible that you have omitted a character between a constant and some identifier, thus fooling the compiler into thinking that the first character of the latter is intended to be a suffix.

### **No Digits After Decimal Point**

You have entered a numeric constant, which contains a decimal point, but you have not entered any digits after that point. Although this is technically a correctly formed number, it has been rejected for reasons of clarity and to ensure that you indeed wish to use a floating point number.

### **Character Constant Too Long**

You have introduced a character constant using the single quote or apostrophe character, but the constant appears to contain more than one character. Perhaps you intended to enter a string constant, or you have neglected to include the closing quote.

### **String Constant Too Long**

You appear to have entered a string constant containing more than 128 characters. Strings greater than this length cannot be handled by this version of the compiler. It is also possible that you have missed off the closing quote, thus causing the compiler to misjudge how much of your code you wish to include in the string constant.

### **Identifier Too Long**

You have entered an identifier that is greater than 30 characters in length, and which is thus beyond the capability of the compiler. It is possible that you did not intend this to be an identifier, and you should check the code around this error for missing operators or other punctuation. If you wanted to enter a string constant, remember to include the opening and closing quotation marks.

### **Invalid Operator**

The compiler has not been able to recognize what appears to be an operator within your code. This error message often appears as a result of general typing errors, which confuse the compiler into thinking you wished to include an operator. You should thus check your code for such mistakes, and for punctuation that is not required in the relevant context.

### **Unexpected Character**

The lexical analyzer has not been able to recognize the character in question as introducing any of the language elements it can process. It is possible that you have tried to use an invalid operator, or that you have made some form of typing mistake. Remember that the “C” and Java operators used by EDICT-97 are different from those used by other programming languages such as Basic.

### **Open Block Comment**

You have started a block comment using the `/*` character sequence, but the compiler cannot find a matching `*/` sequence before the end of the code. This has most likely been caused by your forgetting to include such a sequence, or by starting a comment by mistake.

### **Empty Character Constant**

You have introduced a character constant using the single quote or apostrophe character, but then immediately closed it using the same character, without actually including the character that defined the constant. It is not possible to have an “empty” character constant. If you wish to have such a thing, consider using a space or the value 0 to represent the empty state.

### **New Line in Character Constant**

You have introduced a character constant using the single quote or apostrophe character, but then immediately followed it with a new-line. Character constants cannot be split across line boundaries in this way. This error can be caused by trying to include a backslash in a character constant, and forgetting that such characters should be doubled-up to avoid introducing an escape sequence.

### **New Line in String Constant**

You have introduced a string constant using the double quote character, but then immediately followed it with a new-line. String constants cannot be split across line boundaries in this way. This error is often caused by accidentally omitting the closing quote, causing the compiler to think that the whole of the line in question forms part of the constant. It can also be caused by trying to include a backslash at the end of the string constant,

and forgetting that such characters should be doubled-up to avoid introducing an escape sequence.

### **New Line in Escape Sequence**

You have introduced a so-called escape sequence by using the backslash character within a string or character constant. These sequences are used to include various special values in a constant, and must not be interrupted by line breaks or other formatting characters.

### **Invalid Escape Sequence**

You have introduced a so-called escape sequence by using the backslash character within a string or character constant. These sequences are used to include various special values in a constant, and are formed by following the backslash with one of a number of characters or a numeric value, but the character you have used is not considered valid. This error is most often caused by trying to enter a backslash into a string constant, something that is correctly done by entering a pair of such characters.

### **Invalid Digit in Escape Sequence**

You have introduced a numeric escape sequence by using the backslash character within constant, and then by following that character with “x” or a digit. The compiler has found a digit within the sequence that is not valid in that context. The value in an “x” escape sequence must be two digits of hex, while that in a simple numeric sequence must be three digits of octal.

### **Overflow in Escape Sequence**

You have introduced a numeric escape sequence by using the backslash character within constant, and then following that character with a digit. The compiler has taken the octal value of the following digits, and the value has exceeded the decimal value of 255. Since this is the largest value suitable for inclusion in a string or character constant, this is an invalid sequence.

# Compiler Error Index

This table contains all the error messages, which can be generated by the high-level section of the compiler.

<u><a href="#">Ambiguous Operator</a></u>	<u><a href="#">Array as Operand</a></u>	<u><a href="#">Bit Reference Within Bit</a></u>
<u><a href="#">Bit Reference Within Non-Integer</a></u>	<u><a href="#">Cannot Apply Index Operator</a></u>	<u><a href="#">Cannot Cast an Array</a></u>
<u><a href="#">Cannot Cast to Double</a></u>	<u><a href="#">Cannot Perform Cast</a></u>	<u><a href="#">Empty Sub-Expression</a></u>
<u><a href="#">Expression Has Side Effects</a></u>	<u><a href="#">Expression Too Complex</a></u>	<u><a href="#">Function Argument is Wrong Type</a></u>
<u><a href="#">Illegal Break Statement</a></u>	<u><a href="#">Illegal Case Statement</a></u>	<u><a href="#">Illegal Continue Statement</a></u>
<u><a href="#">Illegal Default Statement</a></u>	<u><a href="#">Illegal Else Statement</a></u>	<u><a href="#">Index Expression is Array</a></u>
<u><a href="#">Index Expression is Non-Integer</a></u>	<u><a href="#">Internal Error [1]</a></u>	<u><a href="#">Internal Error [2]</a></u>
<u><a href="#">Invalid Bit Reference Index</a></u>	<u><a href="#">Invalid PLC Reference</a></u>	<u><a href="#">Missing Function Argument</a></u>
<u><a href="#">Operand is Wrong Type</a></u>	<u><a href="#">Operand Must be Writable</a></u>	<u><a href="#">Operands Not Same Type</a></u>
<u><a href="#">Sign Prefix with Unsigned Constant</a></u>	<u><a href="#">Statement Has No Effect</a></u>	<u><a href="#">Statement Needs Constant</a></u>
<u><a href="#">Statement Needs Integer</a></u>	<u><a href="#">Too Many Function Arguments</a></u>	<u><a href="#">Unexpected Token</a></u>
<u><a href="#">Unknown Function</a></u>	<u><a href="#">Unknown Identifier</a></u>	<u><a href="#">Wrong Number of Arguments</a></u>

## Internal Error

An unexpected error has occurred within the compiler core. This error may be caused by a lack of memory, or by incorrect installation of the software. It may also be caused by a bug within the software, and we would thus ask you to send a copy of the file, which caused the problem to our Technical Support department for study.

## Unexpected Token

The compiler has encountered a token, which it was not expecting, given the current context. The error message will tell you what the compiler has found, and what it was expecting to find. You should examine the code around the position of the error to try and locate the mistake.

## Array as Operand

The operator indicated in the error message cannot accept an array name as an operand. The array name causing the error could be either a single letter representing a Comms block, or the name of global or local data file. This error can be caused by incorrectly formatting the indexing expression that usually follows an array. Remember to use square brackets in such expressions.

### **Operand Must be Writable**

You have used an operator who changes data, but the operand to be changed is not an expression to which a value can be assigned. In order to change a data item, EDICT-97 must be able to resolve the expression in question to a Comms block reference, a variable or the element of a data file.

### **Operand is Wrong Type**

You have used an operator with operands of a type, which the operator is not able to process. For example, you may have tried to subtract two strings, or perform a bitwise operation on a floating point value. You should check the operands to ensure that they are of the type you intended, and you should make sure that the operator itself is correct.

### **Operands Not Same Type**

You have used a binary operator with operands which are not of the same basic type. For example, you may have tried to compare a string and a numeric value, or you may have tried to write a string to a numeric location. EDICT-97 will perform automatic conversions between different sized integers and floating point types, but cannot convert to and from strings. You should check the operands to ensure that they are of the types you intended, and you should make sure that the operator itself is correct.

### **Function Argument is Wrong Type**

You have passed an argument to a function, but the argument is not of the basic data type required. EDICT-97 will perform automatic conversions between different sized integers and floating point types, but cannot convert to and from strings. You should check the arguments of the function to ensure that they are of the types you intended.

### **Function Argument is Wrong Type**

You have passed an argument to a function, but the argument is not of the basic data type required. EDICT-97 will perform automatic conversions between different sized integers and floating point types, but cannot convert to and from strings. You should check the arguments of the function to ensure that they are of the types you intended.

### **Wrong Number of Arguments**

You have invoked a function with the wrong number of arguments. Other less obvious causes of this error message involve using mismatched round bracket within a function argument, or failing to include the final bracket of the function invocation. You should check the definition of the function you are attempting to invoke, and the syntax of the invocation itself.

### **Cannot Apply Index Operator**

You have attempted to use the indexing operator on a data item, which is not a Comms block, a data file or a string expression. The indexing operator is introduced by an opening square bracket, and cannot be applied to any other data types. This error can be caused by using a variable name in place of a Comms block name, or by using a square bracket instead of a round one when attempting to invoke a function.

### **Empty Sub-Expression**

EDICT-97 is attempting to compile a sub-expression within either round or square brackets, but has found the closing bracket instead of any code. This error can occur with parenthesized expressions, with indexing expression, or when using a cast operator. You should check your code and make sure that you have included a valid expression within the brackets.

### **Unknown Identifier**

You have entered a name, which does not correspond to any variable or database item known to EDICT-97. You may already have been offered the chance to create an item to correspond to the unknown name, and you will have declined that offer. This error can also be caused by omitting the round bracket from a function invocation, causing EDICT-97 to assume you are referring to something other than a function. You should check your typing carefully, and make sure you are using a valid name. Remember that EDICT-97 is not case-sensitive as far as identifiers are concerned, and so names differing in case alone will be considered the same.

### **Unknown Function**

You have tried to invoke a function, using a name, which does not correspond to any of the functions known to EDICT-97. This error can also be caused by using a round bracket after an array or Comms block name, in place of the square bracket used to introduce an indexing expression. You should check your typing carefully, and make sure you are using a valid name. Remember that EDICT-97 is not case-sensitive as far as function names are concerned, and so names differing in case alone will be considered the same.

### **Missing Function Argument**

You have omitted one of the arguments of the function you are invoking, either by ending the code after the opening round bracket, or by ending the code after the comma, which follows an argument. As EDICT-97 parses all the arguments in your code before checking that you have supplied the number required by the function, this error can be caused by simply omitting the closing round bracket at the end of a function invocation, even if that function does not need to take any further arguments.

### **Too Many Function Arguments**

EDICT-97 cannot handle function invocations with more than nine arguments. As no EDICT-97 functions need this many arguments, this error will most likely have been caused by omitting the closing round bracket within a function invocation, which forms the argument of another function.

### **Invalid PLC Reference**

You have entered a direct PLC reference, but EDICT-97 cannot parse the string within the square brackets. This error can be caused by using an invalid address for the selected device, by using an invalid device prefix, or by using an invalid type prefix. Remember that some older PLC drivers require an address to be aligned to some fixed multiple in order to be valid. For example, bit data types may have to be aligned to 16-bit boundaries.

### **Sign Prefix with Unsigned Constant**

You have used the unary plus or unary minus operator in front of a constant which has the “U” suffix to indicate that it is unsigned. This contradiction should be resolved by either removing the operator, or removing the suffix.

### **Ambiguous Operator**

You have used the single equals-sign operator, “=”. Although this operator is used within “C” and Java to indicate assignment, EDICT-97 considers the operator too ambiguous, in that many people will attempt to use this operator for both assignment and comparison. You should use “==” for comparing data items, and “:=” for performing assignments operations.

### **Bit Reference Within Non-Integer**

You have attempted to use the bit selection operator with a non-integer left-hand operand. For example, you cannot select bits from within floating point values, from arrays, or from named database items such as display pages. This error can also be caused by including two decimal points within a number, as EDICT-97 will assume the second one is meant to be the bit selection operator.

### **Bit Reference Within Bit**

You have attempted to use the bit selection operator with a left-hand operand that is already a single bit. This will occur if you apply the bit reference operator twice, or to a Comms block element which refers to bit data within the PLC’s memory. This error will not occur in all such circumstances, however, as EDICT-97 will often treat bit data as a 16-bit unsigned value and “forget” about its real size.

### **Invalid Bit Reference Index**

You have attempted to use the bit selection operator with a right-hand operand, which is not an integer. As an example, the value used to select the bit to be examined cannot be a floating point value. This error is normally caused by a simple typing error, such as using a period instead of a comma.

### **Index Expression is Array**

You have used an array name as the indexing expression included within a pair of square brackets. An array name on its own does not make sense in this context, and must be followed by its own indexing expression to select an array element. This error can be caused by using a closing square bracket in place of an opening one, as in "A[A]0]".

### **Index Expression is Non-Integer**

You have used a non-integral value as the indexing expression included within a pair of square brackets. You cannot use a floating point value in this context, or any complex type such as a string or display page name. If you wish to use a floating point value, you must cast it to an integer first.

### **Cannot Cast an Array**

You have attempted to use a type cast sequence on an array name. You can only apply a cast to an individual data item, such as an array element or a variable. This error is most likely to have been caused by a typing error, as operator precedence generally ensures the cast is applied to the correct item.

### **Cannot Cast to Double**

Casting to double-precision floating point is not supported.

### **Cannot Perform Cast**

The cast operation cannot be performed, as conversion from the source data type to the target data type is not supported. For example, you cannot cast a string to a numeric value, or numeric value to a string. In general, casting can be performed between any of the numeric data types, but not between the other more complex types.

### **Illegal Break Statement**

You cannot use the "break" statement except within a loop construct or within the body of a "switch" statement. This error is sometimes caused by the mistaken use of curly braces to group program statements, thus fooling EDICT-97 into thinking that the statement is outside the intended group.

### **Illegal Continue Statement**

You cannot use the “continue” statement except within a loop construct. This error is sometimes caused by the mistaken use of curly braces to group program statements, thus fooling EDICT-97 into thinking that the statement is outside the intended group.

### **Illegal Case Statement**

You cannot use the “case” statement except within the body of a “switch” statement. This error is sometimes caused by the mistaken use of curly braces to group program statements, thus fooling EDICT-97 into thinking that the statement is outside the intended group.

### **Illegal Default Statement**

You cannot use the “default” statement except within the body of a “switch” statement. This error is sometimes caused by the mistaken use of curly braces to group program statements, thus fooling EDICT-97 into thinking that the statement is outside the group within which it should be found.

### **Illegal Else Statement**

You have attempted to use the “else” statement without a corresponding “if” statement. This error is sometimes caused by the mistaken use of curly braces to group program statements, thus fooling EDICT-97 into thinking that the “if” statement has terminated some lines earlier than intended.

### **Statement Has No Effect**

The code you have entered as a statement within the program does not actually do anything. In order to be valid, a statement must either change data, or call a function, which causes EDICT-97 to perform some action. Examples of lines of code which cause this error are “1 + 2;” and “Min(10, 20);”. This error can be caused by including a semi-colon within an expression, thus fooling EDICT-97 into terminating the statement earlier than intended.

### **Statement Needs Constant**

The numeric expression required by the current statement must evaluate to a constant. This error can occur with the “case” statement, which imposes this restriction to allow better optimization of its execution.

### **Statement Needs Integer**

The numeric expression required by the current statement must evaluate to an integer. This error can occur with a loop construct, or with the controlling expression of a “switch” statement. If you need to use a floating point value in any of these contexts, use a type cast to perform the conversion.

**Expression Has Side Effects**

This error occurs when the expression used with a “case” statement changes data or calls a function which performs an action. The restriction whereby all such expressions must be constants is not always sufficient to catch such errors, as an expression like “A[0] := 10” is constant and yet changes data.

**Expression Too Complex**

You have entered an expression, which is too complex to parse. This error occurs when you have too many levels of nested brackets, too many nested function calls, or an expression wherein the operator priorities mandate that many intermediate results are stored for later use. You should simplify the expression, or split it into two sections.

# Section D - Driver Selections

## Communications

EDICT-97's communications architecture is split into three sections...

1. The Comms port table is first of all used to define which Comms drivers are to be used on which of the interface terminal's Comms ports. Most terminals have at least three ports, and so can run up to three Comms protocols at the same time. Note that some drivers, such as those for printers and the like, do not support data communications.
2. The Comms device table is then used to specify the remote devices to be accessed. For simple point-to-point protocols, it is normal to define one device for each Comms driver. For network protocols, any number of devices may be defined, each being identified with a protocol-specific network station address.
3. The Comms block table is finally used to define the data to be transferred between the interface terminal and the remote devices. Each of the 26 blocks can transfer any number of registers between the terminal and a single device, with the direction and rate of transfer being configurable as required.

### Automatic Configuration

If you find the process of setting up the three layers of the Comms structure rather daunting, EDICT-97 contains a mechanism whereby it will automatically configure the necessary items based upon the address entered into the Comms block table. EDICT-97 will offer any address without a corresponding Comms device to every driver available, and then produce a list of those, which are able to handle the address. You can then select the driver you wish to use, and EDICT-97 will automatically bind the driver to a suitable Comms port, and create a default device using that driver.

### Direct PLC References

As well as the Comms block table, EDICT-97 also allows you to enter direct references to data items in remote devices. Such direct references are entered by surrounding the required address in square brackets, with EDICT-97 automatically building its own automatic block table to service these requests. The automatic configuration facility described above also works with direct references, so simply typing something like "[N7:100]" into an expression is sufficient to configure all of the Comms system.

Note that although direct references are the quickest way to get started, explicit Comms blocks allow a greater degree of control, and make it much easier to switch from one Comms driver to another at a later date. Explicit Comms blocks also allow indirect addressing, whereby one data item is used to select an element from a Comms block. This facility is not supported when using direct references.

### Connecting Your HMI to a PLC

Example: Connecting your HMI to a GE Fanuc Series 90.  
(Start with a New File)

### Accessing a Register in your PLC

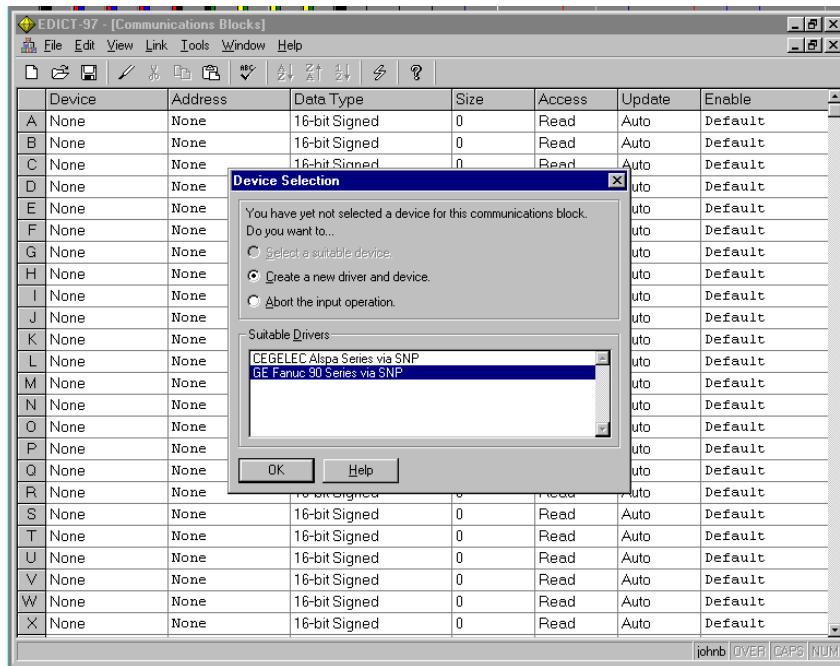
From the GE Fanuc page of Section D, you will see that the Digital Output Register Address starts at %Q0001. Choose **Comms Block** from your Contents Window and the following will appear.

	Device	Address	Data Type	Size	Access	Update	Enable
A	None	None	16-bit Signed	0	Read	Auto	Default
B	None	None	16-bit Signed	0	Read	Auto	Default
C	None	None	16-bit Signed	0	Read	Auto	Default
D	None	None	16-bit Signed	0	Read	Auto	Default
E	None	None	16-bit Signed	0	Read	Auto	Default
F	None	None	16-bit Signed	0	Read	Auto	Default
G	None	None	16-bit Signed	0	Read	Auto	Default
H	None	None	16-bit Signed	0	Read	Auto	Default
I	None	None	16-bit Signed	0	Read	Auto	Default
J	None	None	16-bit Signed	0	Read	Auto	Default
K	None	None	16-bit Signed	0	Read	Auto	Default
L	None	None	16-bit Signed	0	Read	Auto	Default
M	None	None	16-bit Signed	0	Read	Auto	Default
N	None	None	16-bit Signed	0	Read	Auto	Default
O	None	None	16-bit Signed	0	Read	Auto	Default
P	None	None	16-bit Signed	0	Read	Auto	Default
Q	None	None	16-bit Signed	0	Read	Auto	Default
R	None	None	16-bit Signed	0	Read	Auto	Default
S	None	None	16-bit Signed	0	Read	Auto	Default
T	None	None	16-bit Signed	0	Read	Auto	Default
U	None	None	16-bit Signed	0	Read	Auto	Default
V	None	None	16-bit Signed	0	Read	Auto	Default
W	None	None	16-bit Signed	0	Read	Auto	Default
X	None	None	16-bit Signed	0	Read	Auto	Default

Enter the following: **Access (Both), Size ( 3 ), \_Address (%Q0001)**

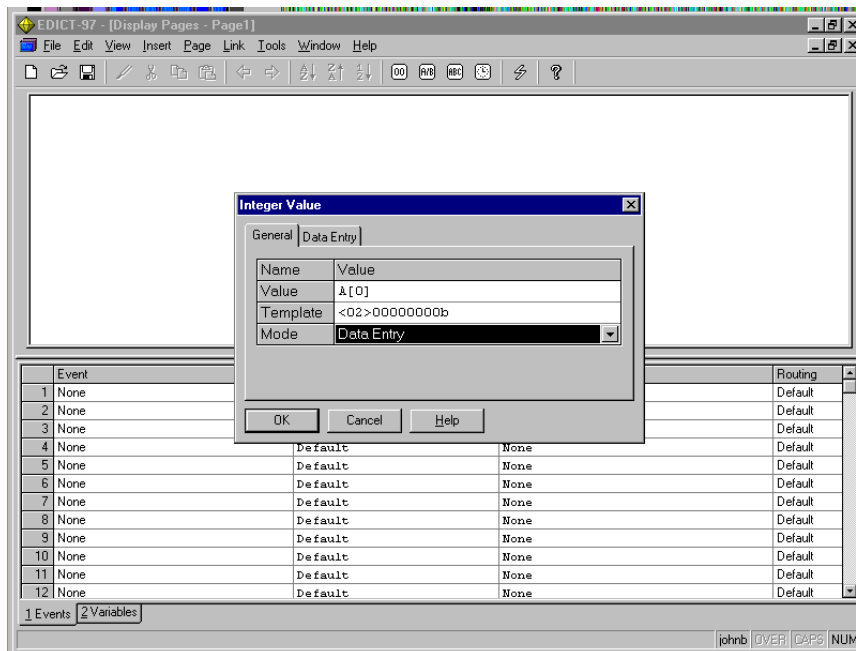
	Device	Address	Data Type	Size	Access	Update
A	None	%Q0001	16-bit Signed	2	Both	Auto
B	None	None	16-bit Signed	0	Read	Auto
C	None	None	16-bit Signed	0	Read	Auto
D	None	None	16-bit Signed	0	Read	Auto
E	None	None	16-bit Signed	0	Read	Auto
F	None	None	16-bit Signed	0	Read	Auto
G	None	None	16-bit Signed	0	Read	Auto
H	None	None	16-bit Signed	0	Read	Auto
I	None	None	16-bit Signed	0	Read	Auto
J	None	None	16-bit Signed	0	Read	Auto
K	None	None	16-bit Signed	0	Read	Auto
L	None	None	16-bit Signed	0	Read	Auto
M	None	None	16-bit Signed	0	Read	Auto
N	None	None	16-bit Signed	0	Read	Auto
O	None	None	16-bit Signed	0	Read	Auto
P	None	None	16-bit Signed	0	Read	Auto
Q	None	None	16-bit Signed	0	Read	Auto
R	None	None	16-bit Signed	0	Read	Auto
S	None	None	16-bit Signed	0	Read	Auto

After you Enter these parameters, the Address %Q0001 triggers the following Window to appear.



%Q0001 is a valid PLC Address for CEGELEC and the GE Fanuc 90 Series. Choose GE Fanuc 90 Series and hit ENTER. EDICT-97 will automatically configure your HMI Communications (Comms Blocks, Comms Ports and Comms Devices) for your GE Fanuc Series 90 PLC.

Return to a Display Page and Enter the following:

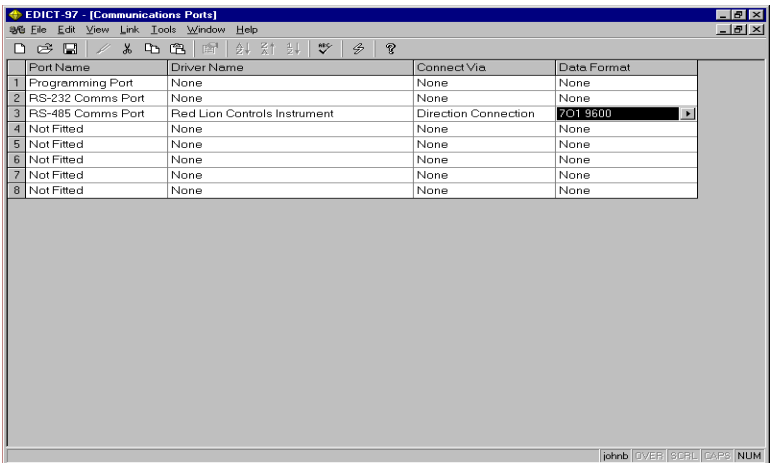


Download the file to your HMI. You should be able to turn the GE Fanuc Series 90 outputs on and off at this time through the front panel.

# Connecting Your HMI to Red Lion Controls products

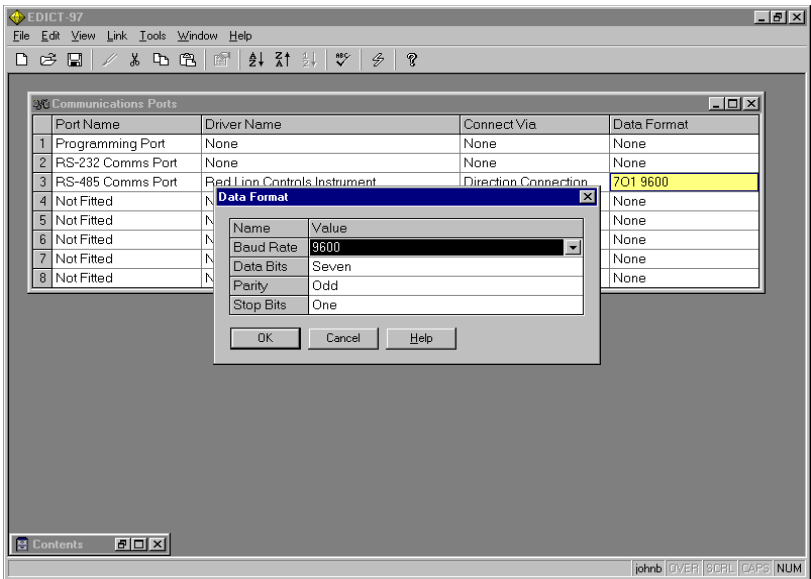
The following simple example shows a Paradigm HMI panel connected to 2 Red Lion Controls' Legend Series Batch Counters (LGB). The Paradigm to RLC RS-485 Connection drawing located later in Section D was used for wiring the 3 units properly.

Configure the Comms Port first.

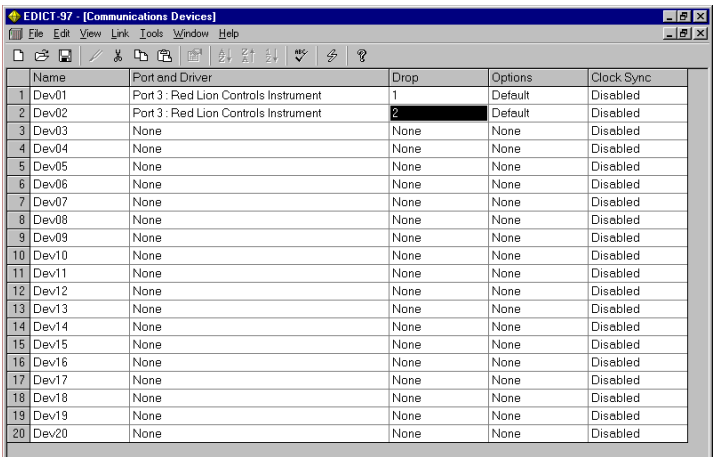


Choose the Red Lion Controls Instrument option from the pull down Driver options. Assign this to the RS-485 Comms Port.

The Communication parameter settings (baud rate, parity, etc.) default settings will match the LGB default settings. If you want to change these settings in the HMI, double click on the **Data Format** Section and the following window appears.



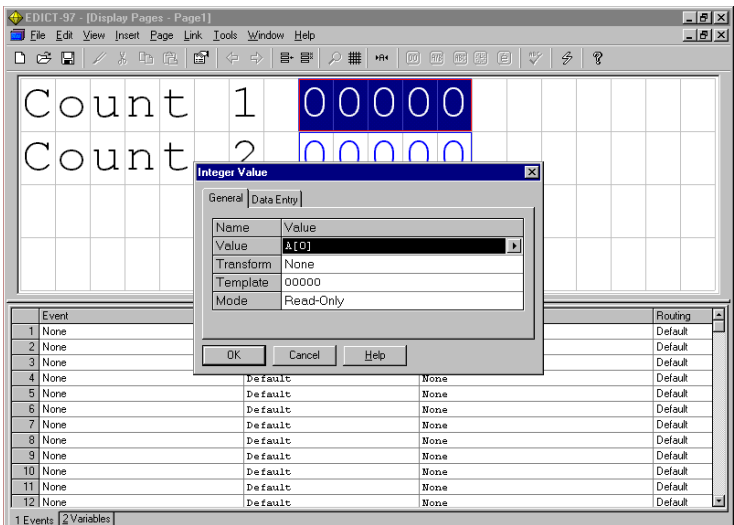
Since you have 2 devices (the 2 LGB's) on the RS-485 of your HMI, you will need to give them different addresses or "Drops". Go to the Comms Devices category and do the following.



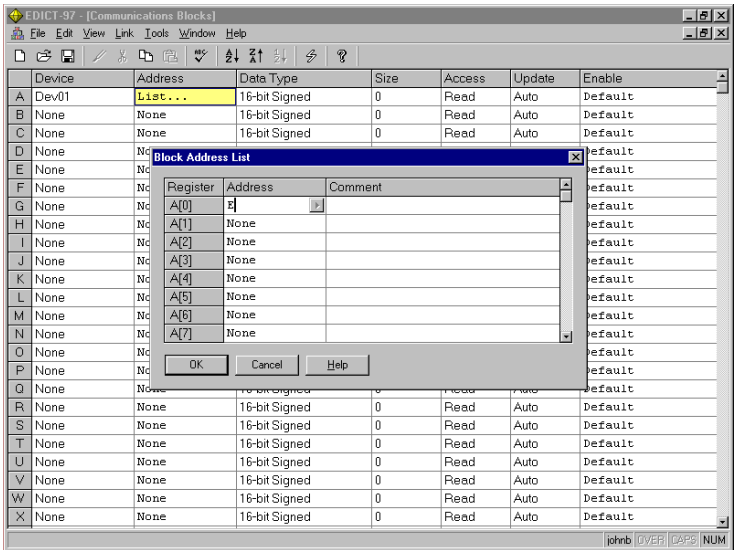
Name	Port and Driver	Drop	Options	Clock Sync
1 Dev01	Port 3: Red Lion Controls Instrument	1	Default	Disabled
2 Dev02	Port 3: Red Lion Controls Instrument	2	Default	Disabled
3 Dev03	None	None	None	Disabled
4 Dev04	None	None	None	Disabled
5 Dev05	None	None	None	Disabled
6 Dev06	None	None	None	Disabled
7 Dev07	None	None	None	Disabled
8 Dev08	None	None	None	Disabled
9 Dev09	None	None	None	Disabled
10 Dev10	None	None	None	Disabled
11 Dev11	None	None	None	Disabled
12 Dev12	None	None	None	Disabled
13 Dev13	None	None	None	Disabled
14 Dev14	None	None	None	Disabled
15 Dev15	None	None	None	Disabled
16 Dev16	None	None	None	Disabled
17 Dev17	None	None	None	Disabled
18 Dev18	None	None	None	Disabled
19 Dev19	None	None	None	Disabled
20 Dev20	None	None	None	Disabled

You can add additional devices easily by giving them different addresses or drops.

To insert the LGB count values into a page on your HMI, do the following.

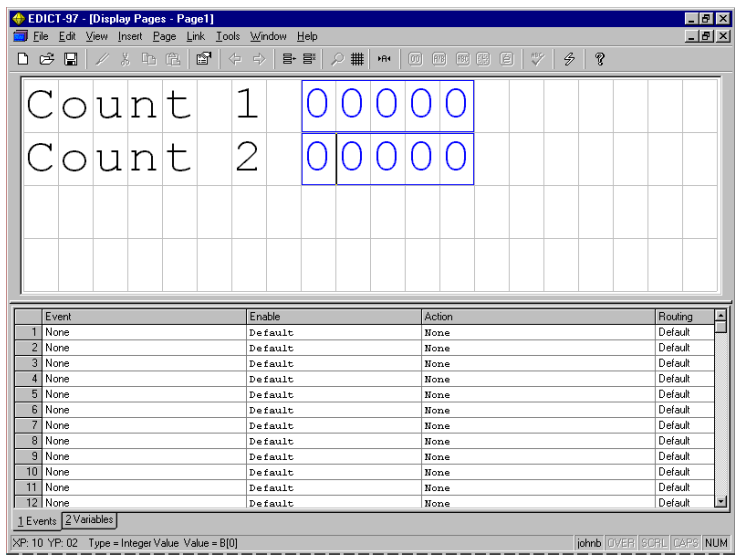
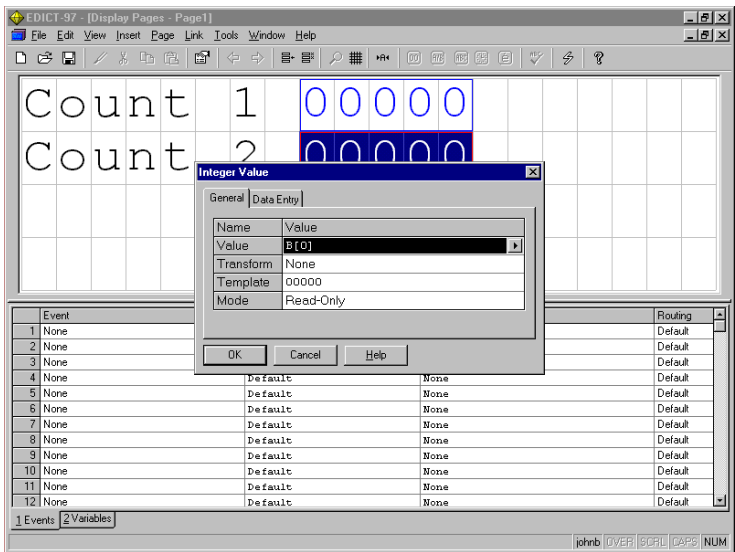


The screenshot shows the 'Display Pages - Page1' window with two 'Count' labels. The first label is followed by '1' and a blue box containing '00000'. The second label is followed by '2' and a blue box containing '00000'. A dialog box titled 'Integer Value' is open, showing the 'General' tab. The 'Value' field is set to 'A[0]', 'Transform' is 'None', 'Template' is '00000', and 'Mode' is 'Read-Only'.



The screenshot shows the 'Communications Blocks' window with a table of communication blocks. The table has columns: Device, Address, Data Type, Size, Access, Update, and Enable. A dialog box titled 'Block Address List' is open, showing a table with columns: Register, Address, and Comment. The 'Register' column contains values A[0] through A[7].

The letter **E** inserted into the address designates that the process value from the LGB is the variable to be inserted into this page. See the **Model LGB Instruction Manual** for additional information on other Value Identifiers.



## Troubleshooting Communications

EDICT-97 has several facilities to troubleshoot communication problems.

### LED's

There are two LEDs(1 green/1 red) on the back of each Paradigm HMI panel.

## State

### **Red LED**

Blinking fast  
Solid on  
Always off

### **Green LED**

Blinking fast  
-  
Blink occasionally

## Status

HMI communicating properly with device  
No Communication link (check wiring between HMI/device)  
Device not understanding HMI (check Baud rate, parity, etc.)

Blink occasionally

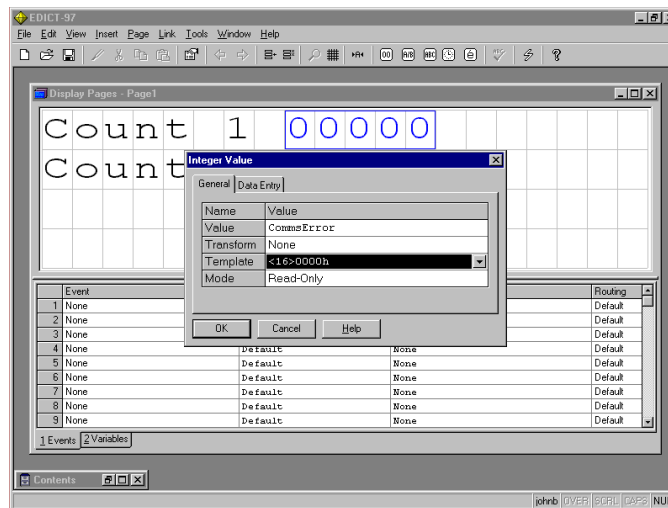
Blink occasionally

Requested Data not available

**The LED's pertain to Ports 2 and 3 on the PLC communication connector (not the programming port)**

## CommsErrors

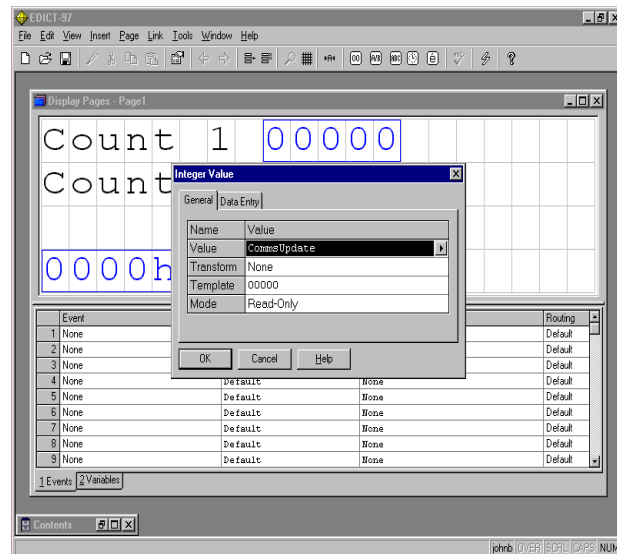
EDICT-97 allows you to insert an Integer value field named CommsErrors on your displayed page which indicates Communication error status. To insert this into a page, do the following.



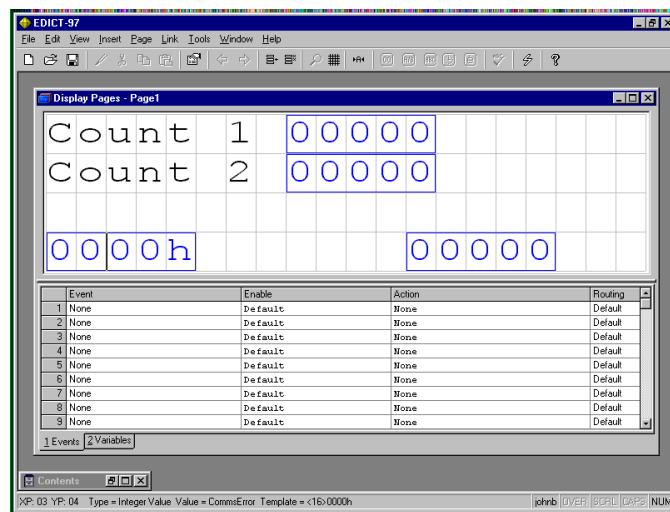
Insert the CommsError Integer value on your display page. Choose the <16>0000h option for the template. If you have proper communication between your HMI and the device the value of 0000h will appear in that Integer field. A CommsError sets the least significant digit to 1 and the device number (drop) that you are having difficulty communicating will set the relevant bit to 1. For example, a CommsError of 0003h indicates that the CommsError is with Device 1; a CommsError of 0005h indicates a CommsError with Device 2, etc. The CommsError Integer value monitors up to 20 devices.

## CommsUpdate

EDICT-97 allows you to insert an Integer value called CommsUpdate to display the communication update in milliseconds. To insert this into your display page, do the following.

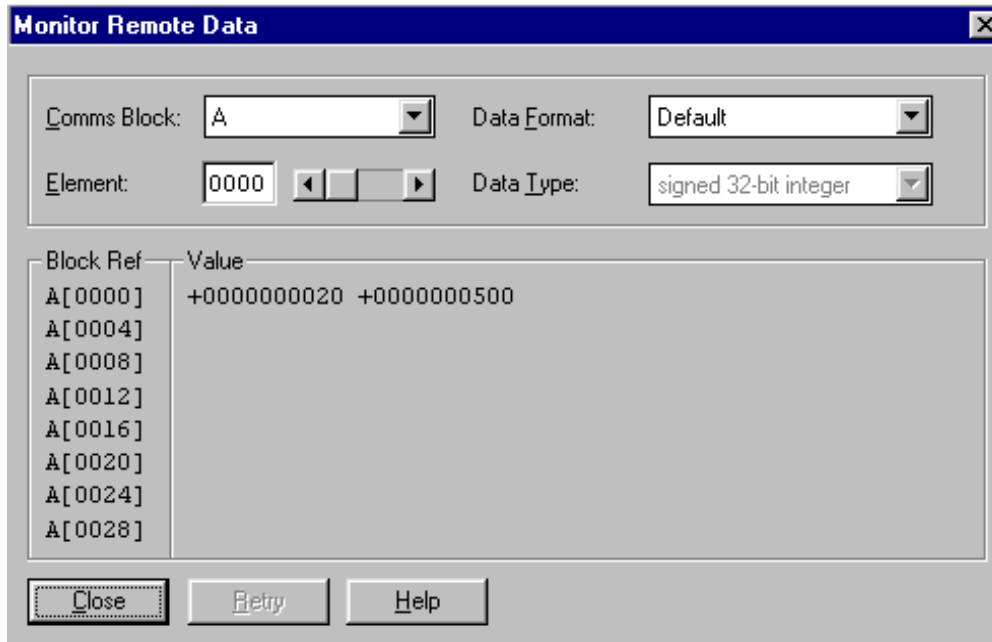


The page below shows a page with both the CommsError and CommsUpdate fields inserted for troubleshooting.



The **Monitor** function can be accessed from the menu line by selecting “Link”, and “Monitor”. This function uses the Programming Port of the terminal to examine the data in the **COMMS BLOCKS**.

The Monitor display can show the data in Decimal, Binary, Octal, and Hexadecimal number systems. It can display the data, depending upon the source, for all the defined data types, i.e. signed, unsigned, string, etc.



The image shows a software window titled "Monitor Remote Data". It contains several input fields and a table of data.

Fields at the top:

- Comms Block: A dropdown menu with "A" selected.
- Data Format: A dropdown menu with "Default" selected.
- Element: A text box with "0000" and navigation buttons (left, right, and a central button).
- Data Type: A dropdown menu with "signed 32-bit integer" selected.

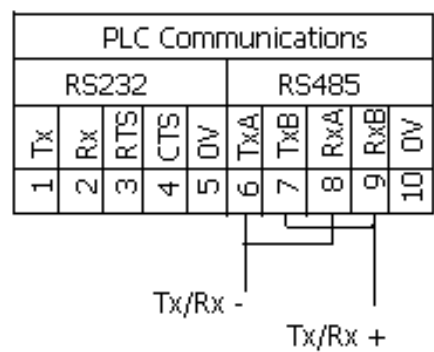
Table below:

Block Ref	Value
A[0000]	+00000000020 +00000000500
A[0004]	
A[0008]	
A[0012]	
A[0016]	
A[0020]	
A[0024]	
A[0028]	

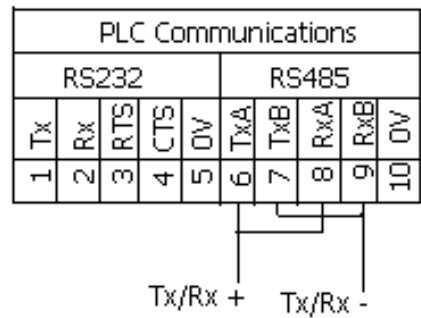
Buttons at the bottom: Close, Retry, Help.

**The Paradigm RS485/422 Port**

To use the RS485/422 Port for RS485 communications; the following wiring applies.



The wiring above is for a Paradigm to DCE type device connection.  
Note: Red Lion Controls products which have RS485 Comm ports are DCE type devices



The wiring above is for a Paradigm to DTE type device connection.

# Paradigm to Paradigm Communications - PCLink

Paradigm units can be configured to pass information to one another using PCLink protocol. They can be used using the RS485/RS422 ports, or daisy-chained using the RS232 ports. The configuration is a master/slave connection, where one unit is the master, and it obtains data from the slave or slaves it is connected to. In a daisy-chain, the units inside the chain are slaved to the previous unit, and a master to the next. One end will be a master only, the other end will be only a slave. For RS485/RS422, one unit will be the master, the others, slaves.

## Configuring a MASTER

### Using the Wizard

The Wizard that you can select when you first enter EDICT97 can configure your MASTER if you want it to. However, most of the time, a user will use the Wizard to configure for the particular PLC attached. Therefore, to additionally configure the PCLink communications, simply use the pictorial presentation that follows.

### Manual Configuration

This is the setup for a MASTER Device on Port 3:

Communications Ports				
	Port Name	Driver Name	Connect Via	Data Format
1	Programming Port	None	None	None
2	RS-232 Comms Port	None	None	None
3	RS-485 Comms Port	Paradigm PCLink Master	Direction Connection	8N1 19200
4	Not Fitted	None	None	None
5	Not Fitted	None	None	None
6	Not Fitted	None	None	None
7	Not Fitted	None	None	None
8	Not Fitted	None	None	None

Then configure your Communications Drivers:

Communications Devices					
	Name	Port and Driver	Drop	Options	Clock Sync
1	Dev01	Port 3 : Paradigm PCLink Master	1	None	Disabled
2	Dev02	Port 3 : Paradigm PCLink Master	2	None	Disabled
3	Dev03	None	None	None	Disabled

In the above case, the Master will use Dev01 to collect data from the slave whose drop number is 1, and Dev02 to collect data from the slave whose drop number is 2.

The data reads and writes will take place according to the way the COMMS BLOCKS are configured:

Communications Blocks							
	Device	Address	Data Type	Size	Access	Update	Enable
A	Dev01	A0	16-bit Signed	1	Read	Auto	Default
B	Dev01	B0	16-bit Signed	1	Write	Auto	Default
C	Dev02	A0	16-bit Signed	1	Read	Auto	Default
D	Dev02	B0	16-bit Signed	1	Write	Auto	Default
E	None	None	16-bit Signed	0	Read	Auto	Default

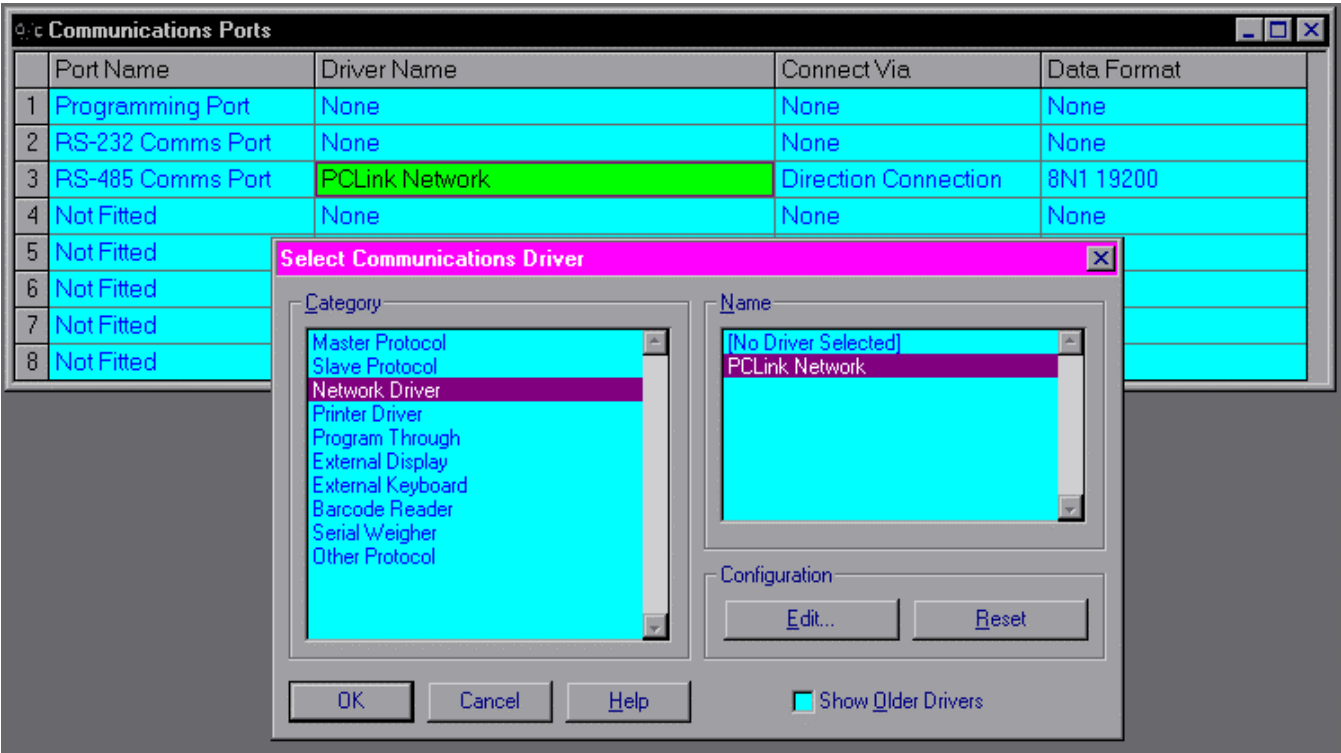
In the above case, the master will read slave 1's Comm Block A[0], and write to slave 1's Comm Block B[0]. It will also read slave 2's Comm Block A[0], and write to slave 2's Comm Block B[0]. If the Size for Comm Block A of the master were 3, and the Address C5, the master would read (or write, as the case may be) C[5], C[6], and C[7], of the slave whose address is 1.

# Configuring the Slaves

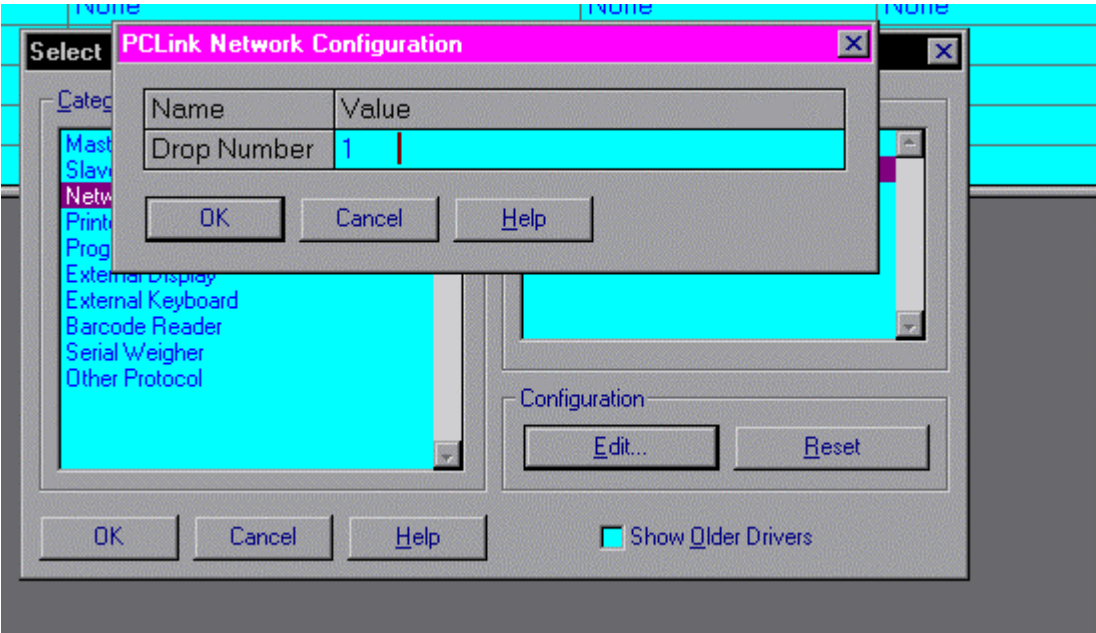
## Using the Wizard

The Wizard that you can select when you first enter EDICT97 can configure your SLAVE if you want it to. However, most of the time, a user will use the Wizard to configure for the particular PLC attached. Therefore, to additionally configure the PCLink communications, simply use the pictorial presentation that follows.


## Manual Configuration



Selecting Edit on the right of the above screen will allow you to define the address of the slave:



You do not need to configure anything in Comms Devices for the slave unit, unless it is attached to a PLC. You do need to have Comms Blocks configure for the registers that the master is reading:

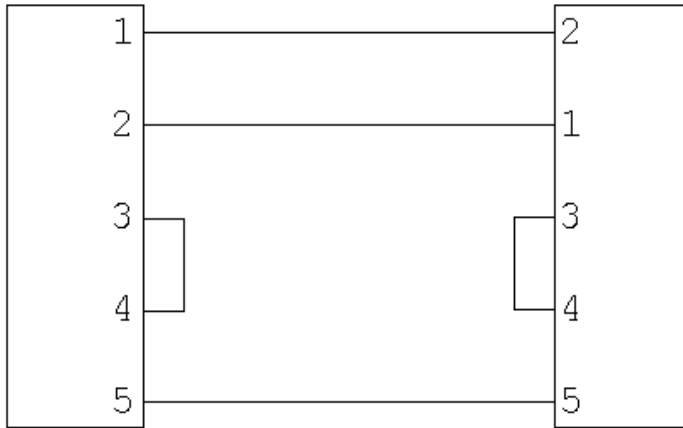
 Communications Blocks							
	Device	Address	Data Type	Size	Access	Update	Enable
A	Internal	None	16-bit Signed	1	Read	Auto	Default
B	Internal	None	16-bit Signed	1	Read	Auto	Default
C	None	None	16-bit Signed	0	Read	Auto	Default
D	None	None	16-bit Signed	0	Read	Auto	Default
E	None	None	16-bit Signed	0	Read	Auto	Default

The above is compatible with the previous programming of the master. By writing values into A[0], the master will have its A[0] (because of its Dev01, address A0) updated.

# Device Connections for PCLink

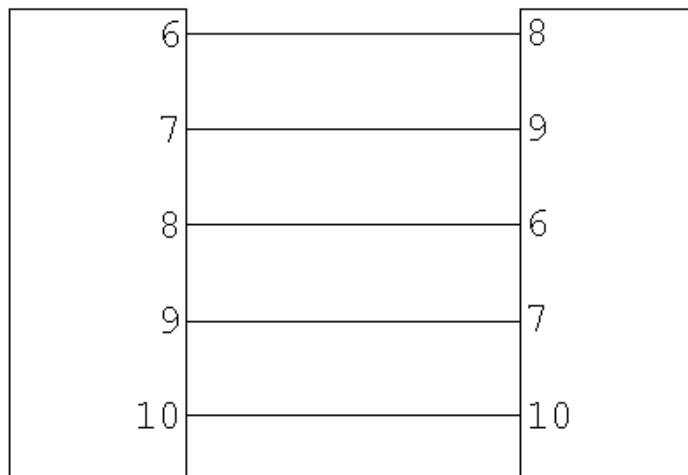
## Using RS-232

Connecting RS-232 is easy. Using Port 2 of both units:



You can get cable number P895803Z to connect a port 2 to a port 1, or use a programming cable (P890301Z) to connect port 1 to port 1.

## Using RS-422



# General ASCII Frame

The General ASCII Frame Protocol is designed to permit communication to any device that transmits and receives ASCII codes. It is capable of receiving ASCII strings, up to 256 characters long, with or without a user-defined start character and/or a terminating character.

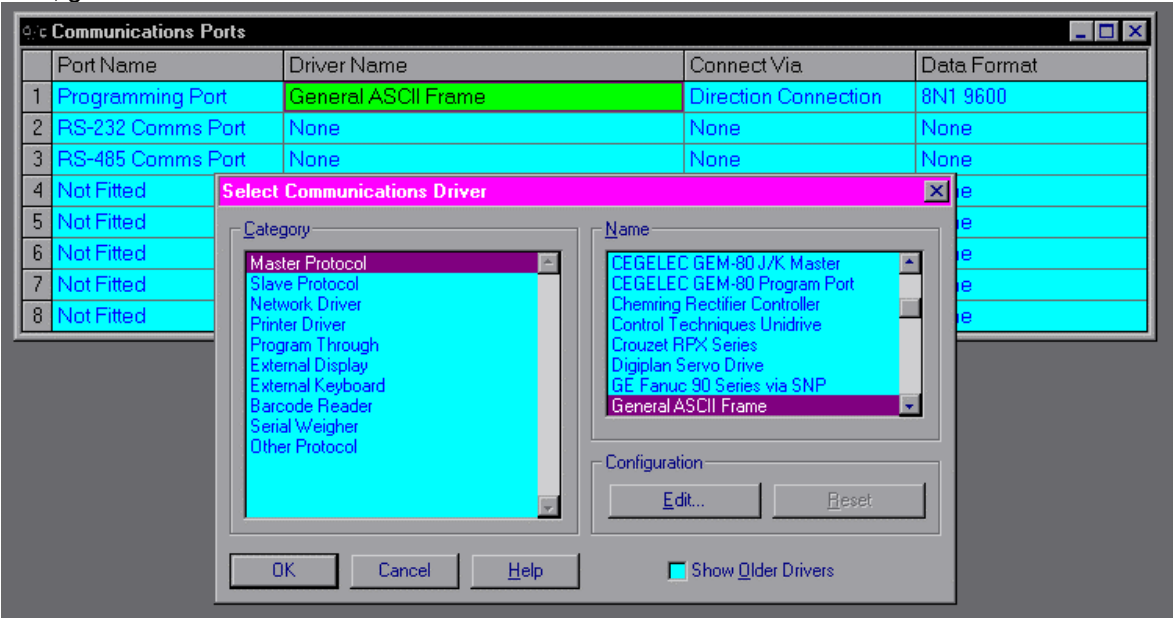
EDICT97 provides many ways of processing the string, from simple display to complex responses.

NOTE: General ASCII Frame is a slave protocol. Therefore it cannot be selected on the same physical port as another driver. On the Paradigm units, Ports 2 and 3 are the same physical port.

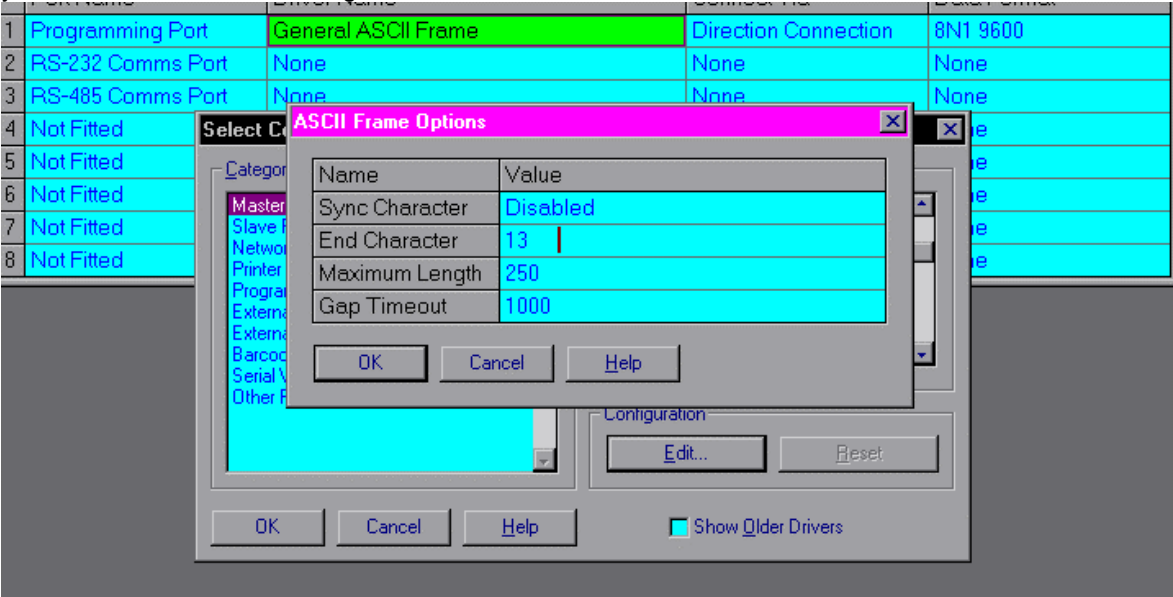
## Using the Wizard

The Wizard that you can select when you first enter EDICT97 can configure your General ASCII Frame protocol if you want it to. However, most of the time, a user will use the Wizard to configure for the particular PLC attached. Therefore, to additionally configure the ASCII communications, simply use the pictorial presentation that follows.

First, go to COMMS PORTS:



Select Edit, in the Configuration section of the window in order to configure the operation the way you need it to work. You will get a window that looks like this:



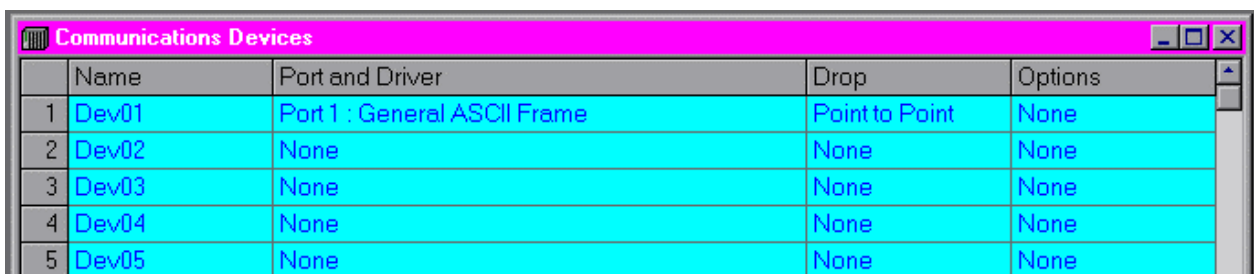
The Sync Character is the Start of Transmission character and is entered as a decimal value. It is not included in the actual string placed in [RXDATA]. When Sync Character is Disabled, the first character received is the first character of the string.

The End Character is the End of Transmission character and is entered as a decimal value. In the above picture, 13 is entered which is the value of Carriage Return. If End Character is Disabled, Maximum Length or Gap Timeout determine the end of the transmission.

Maximum Length will terminate the transmission when that length is reached, regardless of other settings. It does not include the Sync Character.

Gap Timeout will also terminate the transmission, regardless of the other settings. It is defined in milliseconds, from 0 to 9999. The above example shows a one second timeout. If no new character is received within the Gap Timeout, the string will terminate.

Then define the appropriate Device in COMM DEVICES. This defines the internal memory storage for the incoming data. This storage name is represented by [RXDATA]. There is also a counter [RXCOUNT] that counts the number of strings received.



	Name	Port and Driver	Drop	Options
1	Dev01	Port 1 : General ASCII Frame	Point to Point	None
2	Dev02	None	None	None
3	Dev03	None	None	None
4	Dev04	None	None	None
5	Dev05	None	None	None

A simple string display can be accomplished by entering the following on a display page:



The screenshot shows a 'Display Pages - Page1' window with a large display area at the top showing a string of 'G's. Below this, a 'General Text' dialog box is open, showing the configuration for the string display.

Name	Value
Value	[RXDATA]
Length	20
Justify	Left
Mode	Read-Only

Buttons: OK, Cancel, Help

A slightly more complex program might do the following:

Named Data				
	Name	Maps To	Data Type	Transform
1	OldRxCount	Internal	16-bit Unsigned	None
2	LocalCopy	Internal	String [Size 32]	None
3	number	Internal	16-bit Unsigned	None
4	None	Internal	16-bit Signed	None

Global Events			
	Event	Enable	Action
1	Comms update	OldRxCount != [RXCOUNT]	Run (Program1)
2	None	Default	None

User Programs - Program1	
<pre> OldRxCount := [RXCOUNT]; LocalCopy := [RXDATA]; if ( LocalCopy != "" ) {     if ( LocalCopy == "one" )         number := 1;     else if ( LocalCopy == "two" )         number := 2;     else if ( LocalCopy == "three" )         number := 3;     else number := 999; } return; </pre>	

By comparing [RXCOUNT] to a variable OldRxCount in the enable field of the Global Event, the reception of a new string can be detected.

The above three screens, when combined with a General Text display of "LocalCopy" on a Display page, will set "number" equal to one if the word "one" is received, and similarly for "two" and "three". Otherwise, a string will cause number to equal 999. A user could program a statement similar to :

```

if (LocalCopy == "one" )
{
    number := 1;
    gotopage(page1);
}

```

where page one might display a menu for the operator.

**This page intentionally left blank.**

# EDICT-97 Communication Drivers (as of March 04, 2002)

(Drivers are being constantly updated. Contact factory for latest additions.)

---

**ACCU-SORT**    Systems Scanner

---

**ACROLOOP**    Motion Controller

---

**ADAM**    4017-18 Input Module

---

**ALFA LAVAL**

ComLi Master  
ComLi Slave

---

**ALLEN BRADLEY**

PLC-2 via DF1  
PLC-5 via DF1  
SLC via DF1  
SLC via DH485  
DF1 Program Through  
MicroLogix via DF1

---

**ALSTOM**

Alspa Series via SNP  
Alspa via SNP-X Master  
Alspa via SNP-X Slave  
GEM-80 J/K Master  
GEM-80 J/K Slave  
GEM-80 Program Port

---

**ANIMATICS**    SmartMotor

---

**ATLAS COPCO**

DMC  
Programmable Axis Manager

---

**BANNER**    PresencePLUS Sensor

---

**B&R**    Black PLC via CP-6X

---

**BUHLER**    MYEB

---

**CHEMRING**    Rectifier Controller

---

**COMPUMOTOR**    6000 Motor Drive

---

**CONTREX DRIVE**

---

**CONTROL TECHNIQUES**

Mentor II  
Unidrive

---

**CONTROL TECHNOLOGY**    Automation Controller

---

**CROUZET**    RPX Series

---

**DANFOSS**    VLT 6000 Drive

---

<b>DELTA</b>	DVP Series VFD Series (RTU)
<b>DIGIPLAN</b>	Servo Drive
<b>DURANT</b>	Ambassador Counter
<b>ELECTROCRAFT</b>	BRU Master IQ Series
<b>EUROTHERM</b>	Universal Master
<b>EXTERNAL KEYBOARD</b>	ASCII RAFI VT100
<b>FESTO</b>	FPC-Series IPC Series FEC Series
<b>FURNESS CONTROLS</b>	FCO90-2 Detector
<b>GE FANUC</b>	90 Series via SNP
<b>GENERAL ASCII FRAME</b>	(Receives asynchronous input of ASCII Strings)
<b>GIDDINGS &amp; LEWIS</b>	PiC90
<b>HONEYWELL</b>	IPC 620 Series
<b>IAI (INTELLIGENT ACTUATOR)</b>	Super SEL Controller
<b>IDEC</b>	FA-3S via FA-1 (PF2-CLA Link Adapter) Micro 3 Series
<b>IFM</b>	AS-I Programming Interface
<b>IMO</b>	G Series PLC Jaguar CD Drive Jaguar VX Drive Jaguar VX-S Drive K Series PLC Nexus Series
<b>INDRAMAT</b>	CLC Motion Control SYNAX via 3964R
<b>KEB</b>	Frequency Inverter
<b>KEYENCE</b>	KV Series
<b>KLOCKNER MOELLER</b>	PS-306 PS4-201
<b>LENZE DRIVE</b>	via LECOM via LECOM Type II

---

**MATSUSHITA / AROMAT**

FP Series  
Program Through

---

**MICROMO MVP Motion Controller**

---

**MINAS** Series AC Servo Drive

---

**MITSUBISHI**

A Series via A1SJ71C24-R2 Communication Module  
F2 Series  
FX Series  
FX Series Program-Through

---

**MODBUS MODICON**

ASCII Master  
ASCII Slave  
RTU Master  
RTU Slave  
Universal Master  
Extended Universal Master

---

**OMRON**

Sysmac C-Series  
Sysmac Program Through

---

**OPTIMISED CONTROL MINT**

---

**PACIFIC SCIENTIFIC**

OC930  
PC830 Servo Drive

---

**PARKER** 6K Series Motion Controller

---

**PCLINK MASTER** (Master for Paradigm to Paradigm communications)

---

**PCLINK SLAVE** (Slave for Paradigm to Paradigm communications)

---

**PLC DIRECT**

Koyo via DirectNET  
Koyo via K-Sequence

---

**PROFIBUS** Network

---

**QUICKSILVER** SilverMax

---

**RED LION CONTROLS INSTRUMENTS**

---

**RED LION CONTROLS INSTRUMENTS via MODBUS**

---

**ROLL-YOUR-OWN-PROTOCOL**

---

**SABROE** COMSAB II Module

---

**SIEI DRIVE** via SLINK3

---

**SIEMENS / SIMATIC / TEXAS**

S7 via FreePort  
S7 via MPI Adapter  
S7 via PPI  
TI-325 and TI-330  
TI-500 Series

---

---

**SIMOVERT** P via USS

---

**SIMPLE ASCII PRINTER**

---

**SPRECHER & SCHUH** SESTEP

---

**SQUARE-D** SYMAX Series

---

**TELEMECANIQUE** UNI-TELWAY

---

**TOSHIBA**

EX Series

G3 Tosvert-130 Inverter

T2 Series

---

**UNITRONICS** M90

---

**WEST INSTRUMENTS** 6100

---

**WMCU CONTROLLER** Weather Monitoring & Control Unit

---

**YASKAWA**

MP Series Controller

Sigma II Indexer (JUSP-NS600)

---

04 March 2002

# Paradigm Cable Guide (as of March 04, 2002)

(Contact factory for latest additions.)

---

## PARADIGM

- P890301Z (Supplied with SFEDT Development Kit) RJ-11 - RS232 for programming the Paradigm with PC or for RS232 to 9 pin female connection to Paradigm Port 1
- P890301C Cable Adapter, RJ-11 Jack to DB-9 (cable not included)
- P890301J RJ-11 to RJ-11 (used for communications between P890301J Paradigm's Port 1 and the Profibus Host Adapter, PAPBH000)
- P895803Z PCLink, Paradigm Port 1 to Paradigm RS232 port (unit to unit programming link)
- P890806Z Cable Adapter, Paradigm Port 1 RJ-11 to any 10-position RS232 Communication Cable
- P890807Z Cable Adapter, Paradigm Port 1 RJ-11 to any 5-position RS232 Communication Cable
- P890808Z Cable Adapter, Paradigm Port 1 RJ-11 to any 10-position RS232 Communication Cable, DIN Rail Mount

---

## ALLEN BRADLEY

- P895005Z SLC500 via DF1 (RS232 9 pin Female)
- P895006Z PLC-5 via DF1 (RS232 25 pin Male)
- P895007Z PLC-2 via DF1 (RS232 15 pin Male)
- P895013Z SLC500 via DH485 (DH485 RJ-45)
- P895047Z Micrologix (Series C and higher) via DH485 through 1761-CBL-PM02 (RS232 9 pin Male)

---

## BANNER

- P895064Z PresencePLUS Sensor

---

## CEGELEC (Alstom)

- P895016Z Alsipa 80-35 (RS422 15 pin male)

---

## DELTA

- P895063Z DVP Programming Port

---

## GE FANUC

- P895015Z Series 90 (RS422 15 pin Male)
- P895057Z CMM311 (RS422 25 pin Male)

---

## GENERIC

- P895047Z RS232 9 pin Male
- P895058Z RS232 to bare wires
- P895059Z RS422/RS485 to bare wires

---

## HONEYWELL

- P895027Z IPC620 (RS422 25 pin Male)

---

## IDEC

- P895028Z FA Series via FA-1 (RS232 25 pin Male)
  - P895045Z Micro 3 (RS485 8 pin Male Mini-DIN)
  - P895061Z Micro3C (RS232 8 pin Male Mini-DIN)
-

**IMO**

P895049Z Nexus (RJ-45)

---

**KEYENCE**

P895018Z KV Series (RS232 RJ-11)

---

**MATSUSHITA / AROMAT**

P895031Z FP Series (RS232 9 pin Male)  
P895062Z FP0  
P895047Z FP0 via Programming Cable  
P895047Z FP1 Series via AFP15201-US9 cable (RS232 9 pin Male)

---

**MITSUBISHI**

P895002Z A Series via A1SJ71C24-R2 Card (RS232 9 pin Male)  
P895033Z A Series (RS422 - Wire Connection)  
P895003Z F2 Series (RS232 25 pin Male)  
P895001Z FX Series (RS422 25 pin Male)  
P895004Z FX0 & FX0N (RS422 8 pin Male Mini-DIN)

---

**MODBUS MODICON** See also Square-D and Telemecanique

P895019Z RS232 9 pin male  
P895060Z TSX Micro, Nano, Premium (RS485 8 pin Male Mini-DIN)

---

**OMRON SYSMAC**

P895021Z C200H or LK201 (RS232 25 pin male)  
P895047Z CP Series through CQM1-CIF02 Cable (RS232 9 pin male)  
P895047Z CQ Series through CQM1-CIF02 Cable (RS232 9 pin male)  
P895052Z CQ Series ( RS232 9 Pin male )  
P895052Z CP Series through CPM1-CIF01 Adapter (RS232 9 pin male)  
P895052Z C200HS (RS232 9 pin male)

---

**OPTIMISED CONTROL**

P895022Z MINT (RS422 9 pin female)

---

**PLC DIRECT**

P895030Z Koyo 205 (RS232 RJ-11)  
P895046Z Koyo 405 via RS232(RS232 25 pin Male)

---

**RED LION CONTROLS**

P893805Z MODBUS RS485 (RJ-11)

---

**SCADA**

P895026Z to PC

---

**SIEMENS / SIMATIC / TEXAS**

P895012Z CP525 SI (RS422 15 pin Male)  
P895046Z TI435 (RS232 25 pin male)  
P895047Z S7 Direct connection (RS232 9 pin male)  
P895048Z S7 through PC/PPI Programming Cable (RS485 9 pin male)  
P895053Z S7 via MPI Adapter (included with cable) (RS232 9 pin male)  
P895053Y S7 via MPI Adapter (MPI not included) (RS232 9 pin female)  
P895055Z 545/555 via RS232 (9 pin female)  
P895056Z 545/555 via RS422 (9 pin female)

---

**SPRECHER & SCHUH**

P895017Z 390 (RS232 15 pin male)

---

**SQUARE-D**

P895008Z Symax Series via Programming Port (RS422 9 pin male)

---

**TELEMECHANIQUE**

P895023Z TSX (RS485 15 pin male)

---

**TEXAS INSTRUMENTS**

P895024Z 545 (RS232 9 pin male)

P895043Z 545 (RS422 9 pin male)

---

**TOSHIBA**

P895009Z EX100 and M Series (RS422 to wire connection)

P895010Z T2 (RS422 15 pin male)

P895011Z T1 (RS232 8 pin Mini-DIN)

---

**YASKAWA**

P895054Z MP930 (RS232 9 pin male)

P895065Z JUSP-NS600 RS232 3M

P899065Z JUSP-NS600 RS232 custom length

P895066Z JUSP-NS600 RS485 3M

P899066Z JUSP-NS600 RS485 custom length

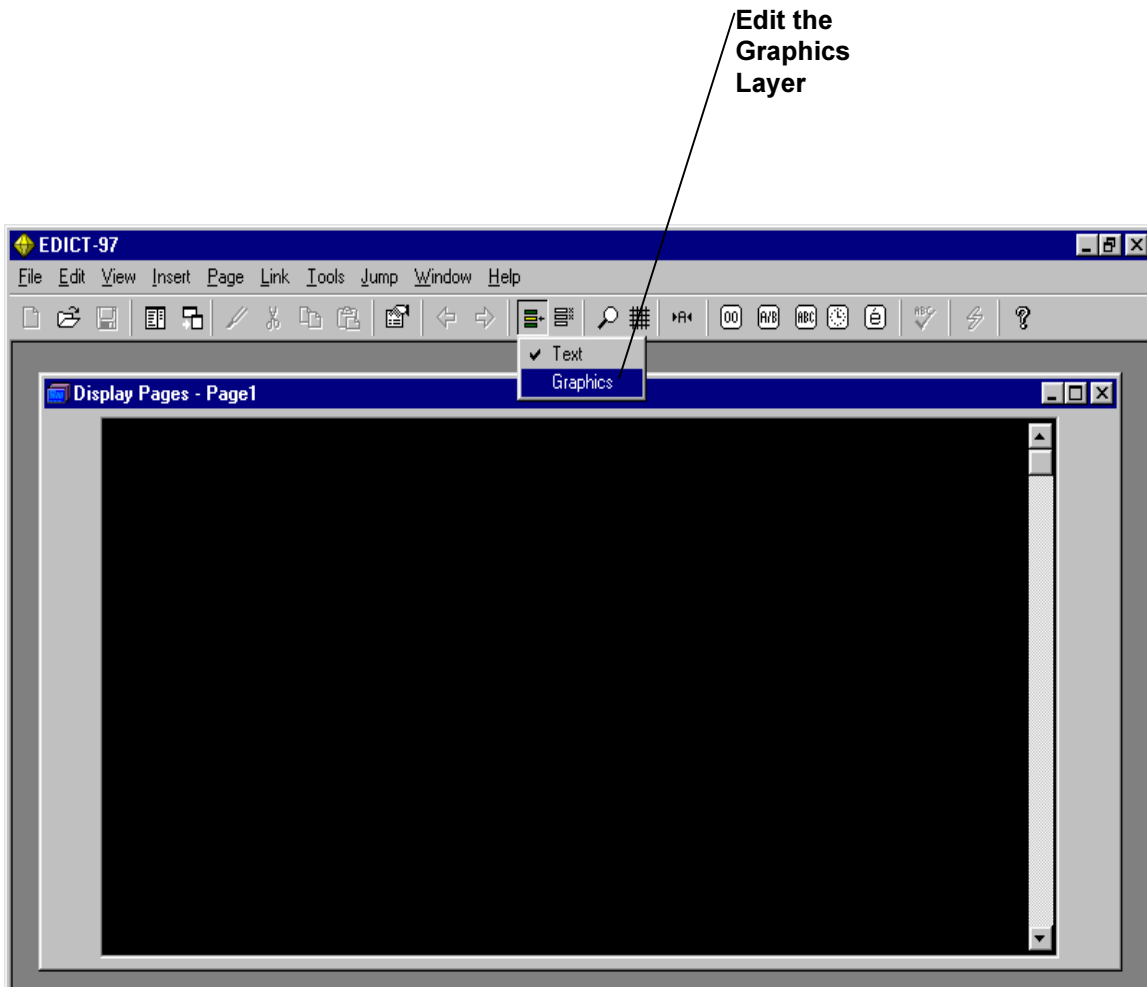
---

04 March 2002

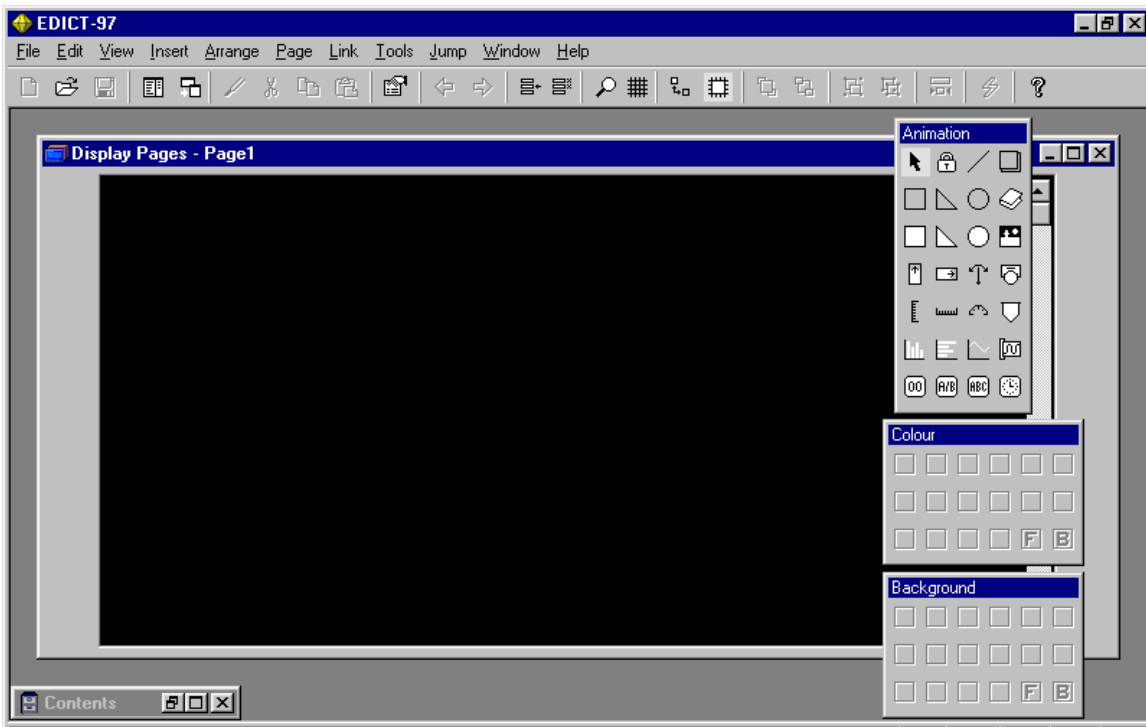
## Section E (Graphic Units)

In addition to all the features of the character-based units, the Graphic units will provide exceptional value in displaying trend graphs, process schematics and flow, and others, limited only by the imagination of the designer. Color Graphic units support 16 color Animation objects, which adds to the flexibility of page design. EDICT-97's extensive Library of predefined objects along with the ability to create custom animation macros makes page creation easy. Color Graphic units also feature dynamic PowerPoint type page transitions. These transitions can add to realistic representations of machines, large plants, etc.. Touch units feature the ability to make any object placed on a page a touch sensitive input. Any of EDICT-97's Actions (See The Action Builder, Page 32) can be programmed to occur once an object is pressed, maintained or released.

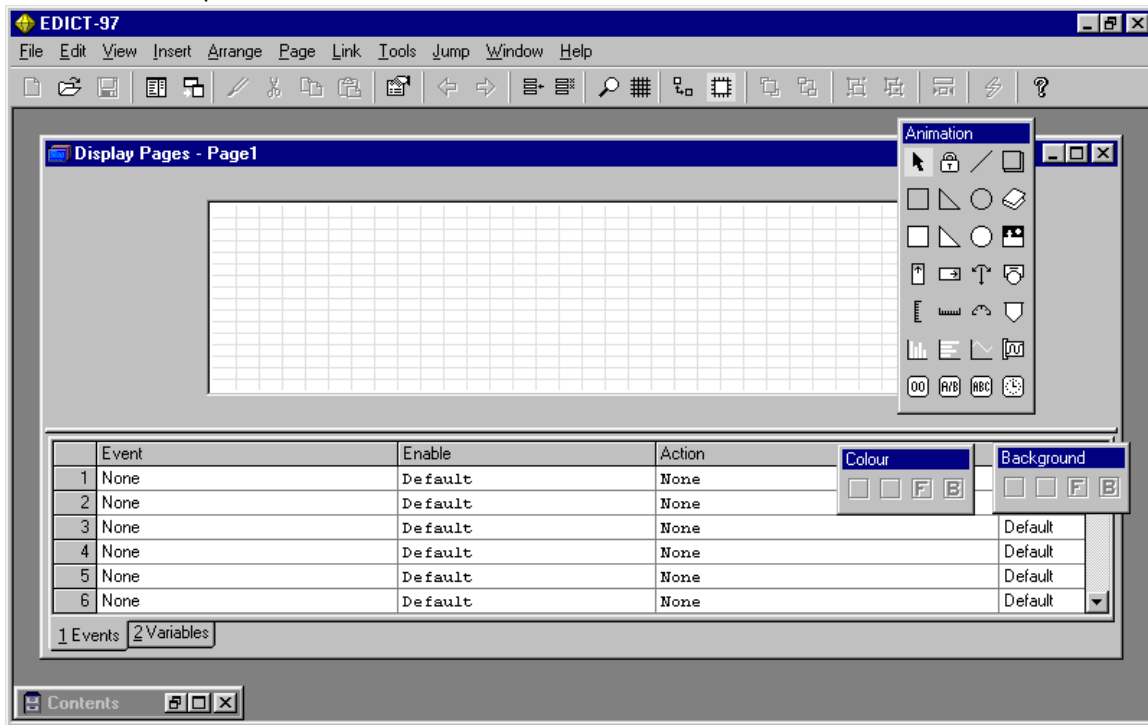
### Accessing the Graphics Layer



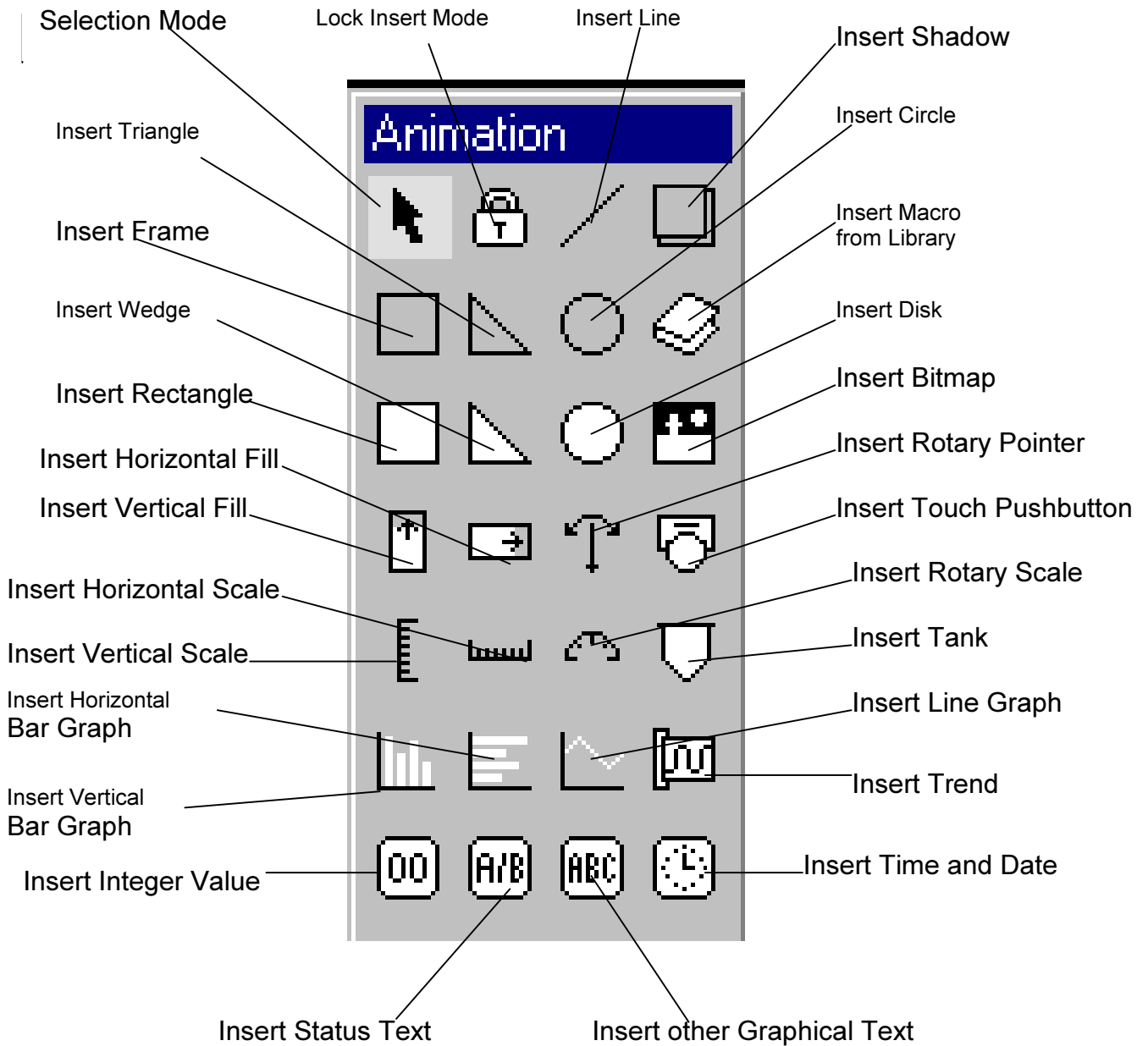
The following ToolBoxes appear  
Color Graphic Unit



Monochrome Graphic Units



# The Graphics Animation Toolbox



## Items from the Animation ToolBox

The tables below list the various animation items that EDICT-97 supports...

### Values and Text

<b>Name</b>	<b>Description</b>	<b>For Graphic Layer Properties see Page</b>
<u>Integer Value</u>	Displays an integer value	E49
<u>Real Number</u>	Displays a fixed point real number.	E50
<u>Fixed Text</u>	Displays a fixed text string.	E52
<u>Status Text</u>	Displays one of two messages.	E53
<u>Quad Text</u>	Displays one of four messages.	E54
<u>Message Text</u>	Displays a message from a numbered list.	E56
<u>Decode Text</u>	Displays a message from a condition list.	E58
<u>General Text</u>	Displays a general string expression.	E59
<u>Time &amp; Date</u>	Displays the current time and/or date.	E60

### Basic Figures

<b>Name</b>	<b>Description</b>	<b>For Graphic Layer Properties see Page</b>
<u>Line</u>	A line between two points.	E61
<u>Frame</u>	A rectangular frame.	E62
<u>Rectangle</u>	A solid rectangle.	E62
<u>Shadow</u>	A rectangular frame with a drop shadow effect.	E63
<u>Wedge</u>	A solid triangle within a defined rectangle.	E63
<u>Touch Push Button</u>	From EDICT-97's object library.	E19
<u>Circle</u>	An outline circle.	E63
<u>Disk</u>	A solid circle.	E64

### Fills

<b>Name</b>	<b>Description</b>	<b>For Graphic Layer Properties see Page</b>
<u>Horizontal Fill</u>	A horizontal fill based upon some data value.	E20
<u>Vertical Fill</u>	A vertical fill based upon some data value.	E20

## Graphs

<b>Name</b>	<b>Description</b>	<b>For Graphic Layer Properties see Page</b>
<u>Horizontal Graph</u>	A graph of horizontal bars based upon a data list.	E29
<u>Vertical Graph</u>	A graph of vertical bars based upon a data list.	E29
<u>Line Graph</u>	A line graph of points based upon a data list.	E34
<u>Rotary Pointer</u>	A rotary line based upon a value. ( 9 different styles).	E24
<u>Data Trend</u>	A line graph showing a Data Recorder channel.	E38

## Scales

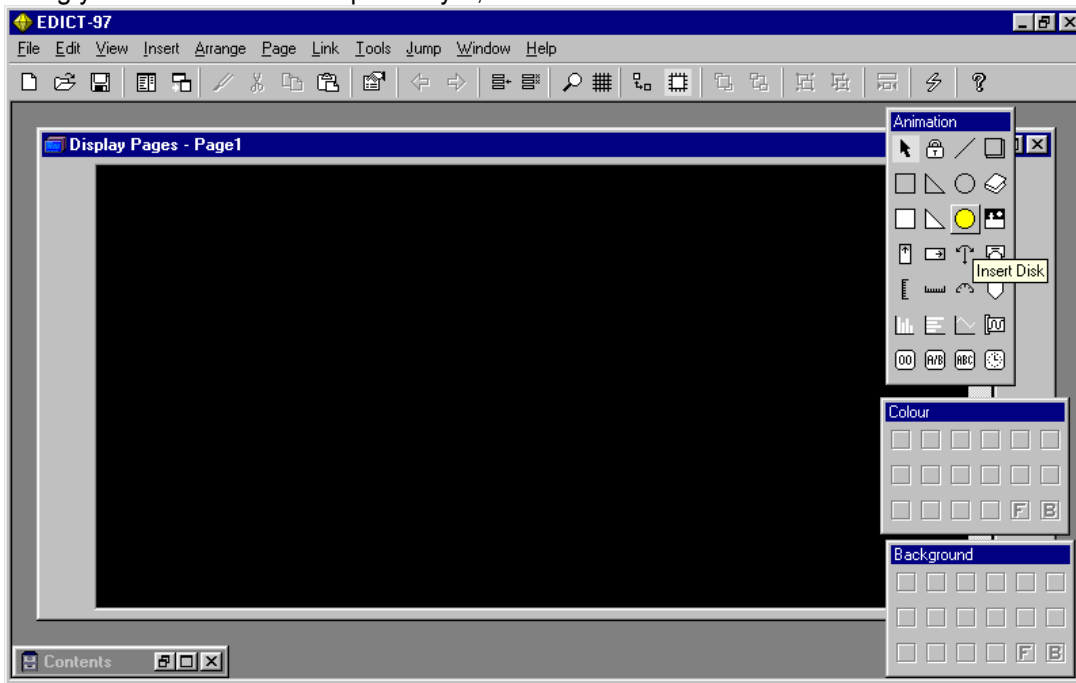
<b>Name</b>	<b>Description</b>	<b>For Graphic Layer Properties see Page</b>
<u>Horizontal Scale</u>	A horizontal “comb” used to label an axis.	E23
<u>Rotary Scale</u>	A rotary “comb” used to label the rotary pointer style.	E27
<u>Vertical Scale</u>	A vertical “comb” used to label an axis.	E23

## Others

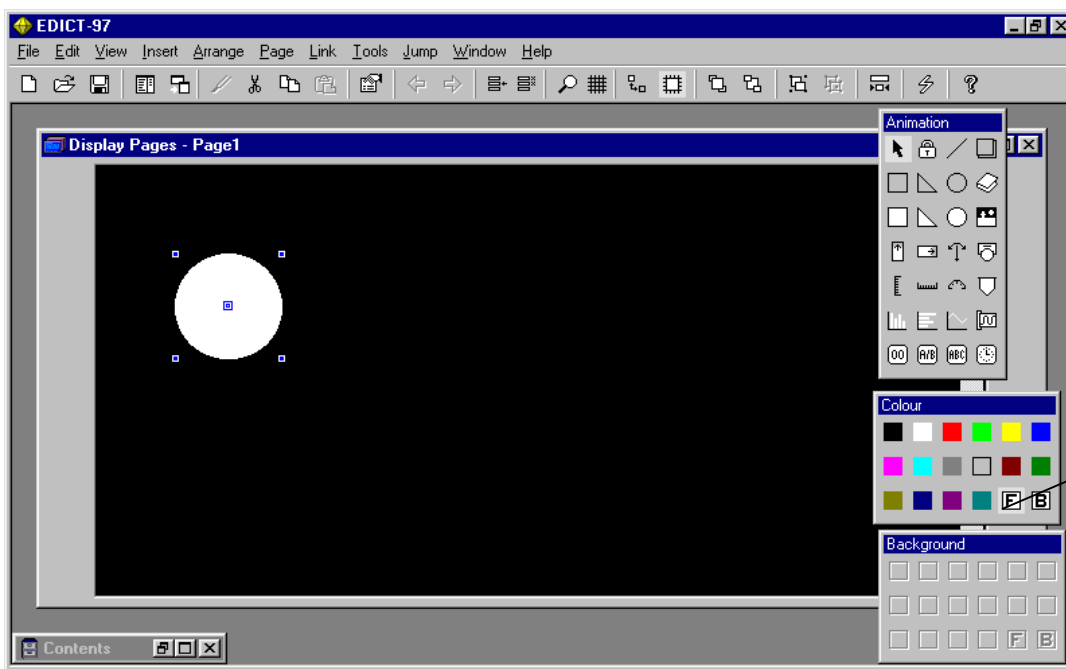
<b>Name</b>	<b>Description</b>	<b>For Graphic Layer Properties see Page</b>
<u>Bitmap</u>	A bitmap from the Bitmap Images list.	E45
<u>Macro from Library</u>	A predefined object from EDICT-97/ User’s Library.	E19

## Inserting animation items on a display page for a VX500T (color touch screen)

Using your mouse on the Graphics layer, select **Insert Disk**.

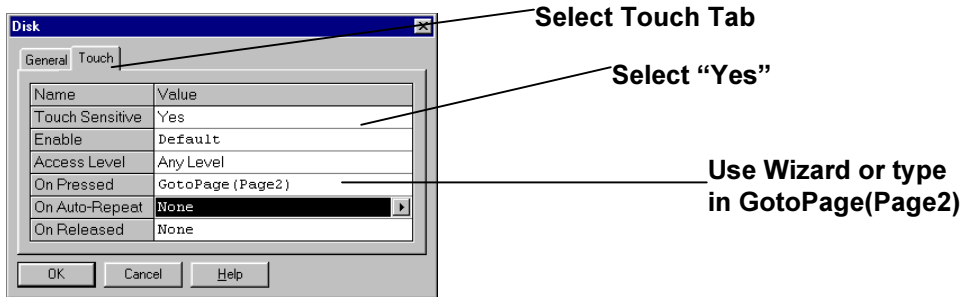


Using your mouse, move your cursor to the area on the page where you want to insert the disk. Hold your left mouse button down and drag the disk to the desired size. Now release the left mouse button.

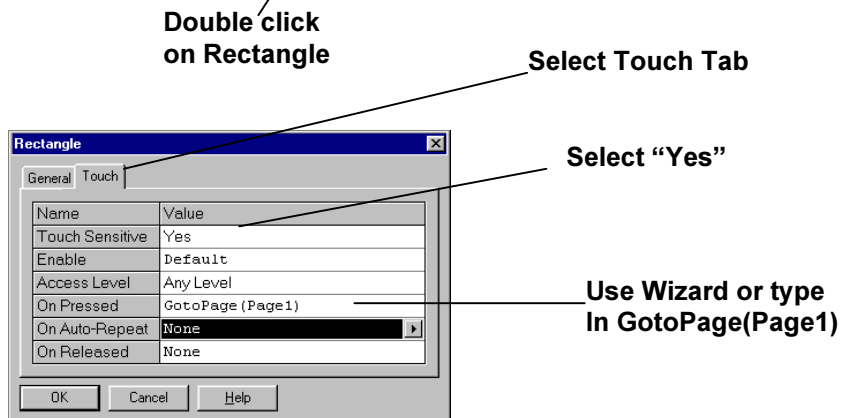
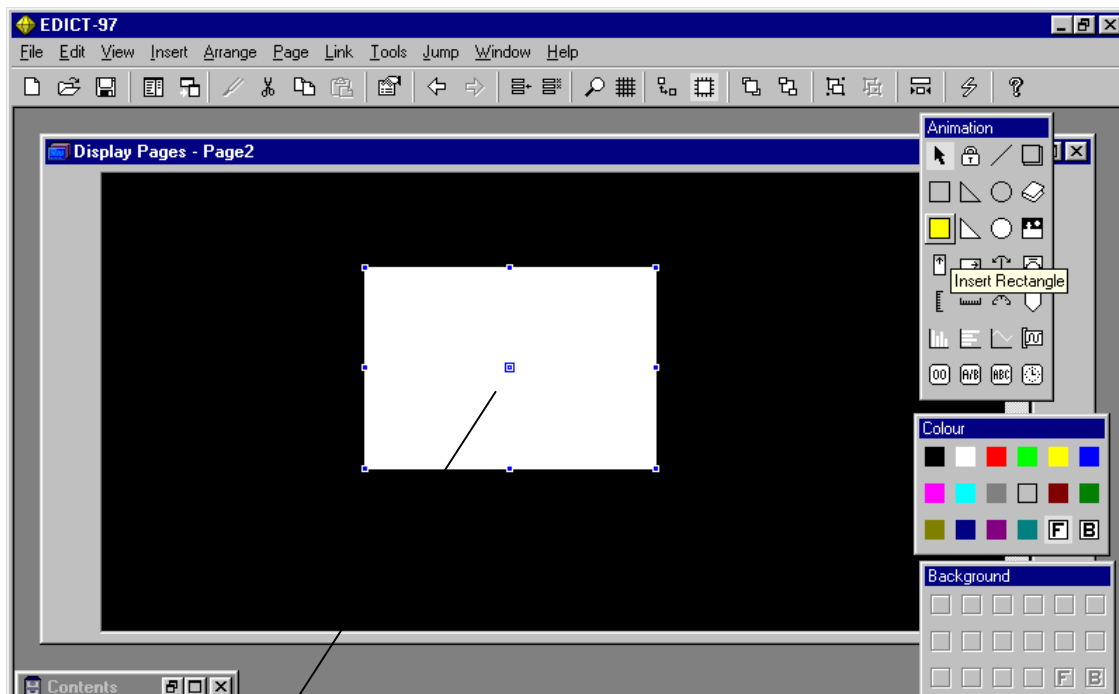


To change color of disk, click **Foreground**. Then click desired color.

Move your cursor to the disk on the page. Now double click the disk to edit the disk's properties.



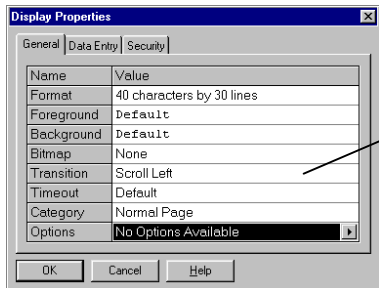
Now go to Page 2 and repeat the same process. This time use a rectangle.



Download this database to your VX500T. Touching the disk and rectangle will toggle your display between Page 1 and Page 2.

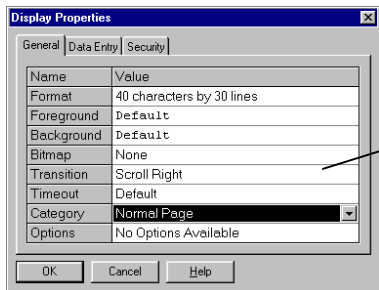
## PowerPoint-style Page transitions(color units only).

Using the same database, Go to Page1 and select **Page/Properties** from the top of the page.  
The following window appears.



Select "Scroll Left" for transition

Click OK and go to Page2.  
Select **Page/Properties** and the following window appears.

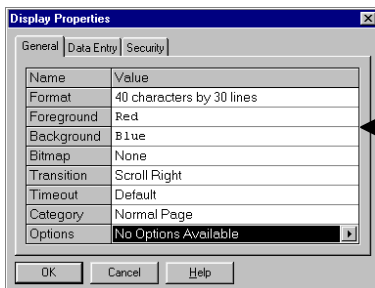


Select "Scroll Right" for transition

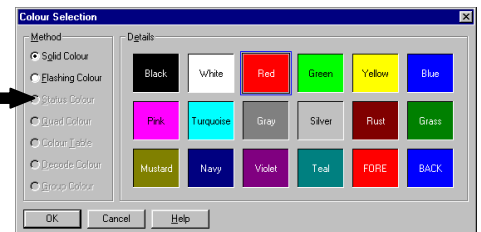
Click OK and download to your VX500T and touch the disk and rectangle to observe the dynamic page transitions. To observe how the other page transitions behave (Split Horizontal,etc.) go back to the **Page/Properties** windows and make alternate selections.

## Choosing a Display Page color.

Using the same database; select **Page/Properties** of Page2.



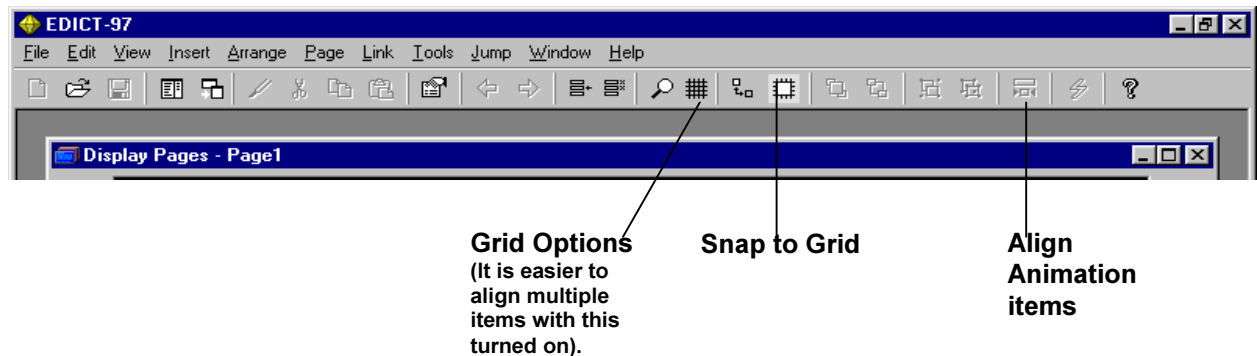
Expand the Foreground and Background windows to choose a color from the Color Selection window.



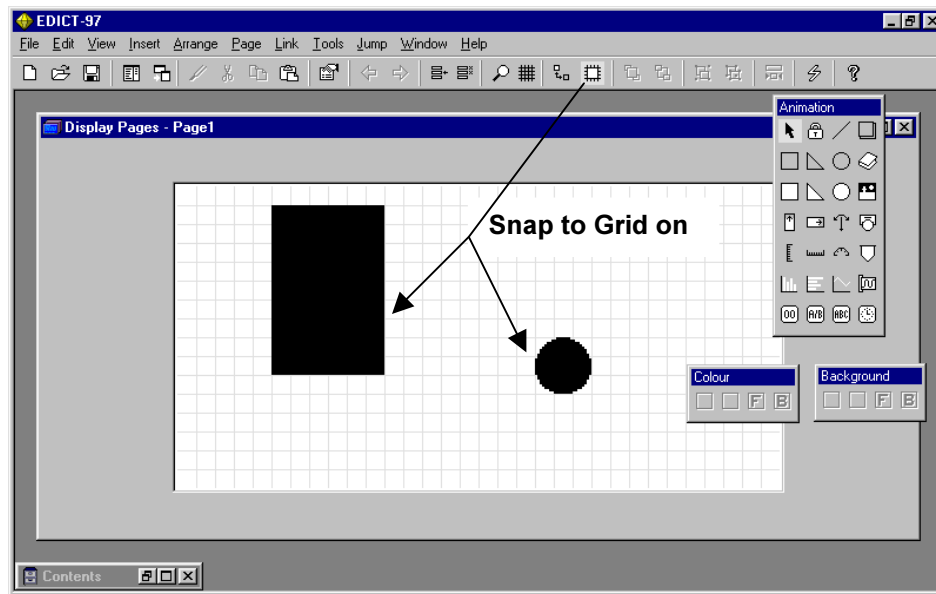
Background color refers to the display page background color. Foreground color refers to the default color that animation items will be unless otherwise specified. Both Background and Foreground colors can be either solid or flashing colors.

## Inserting multiple animation items on a display page.

The following Icons appear at the top of the Graphics layer window.



This example shows how to align a rectangle and a disk inserted on the Graphics layer of a Monochrome Graphic unit (GL350).

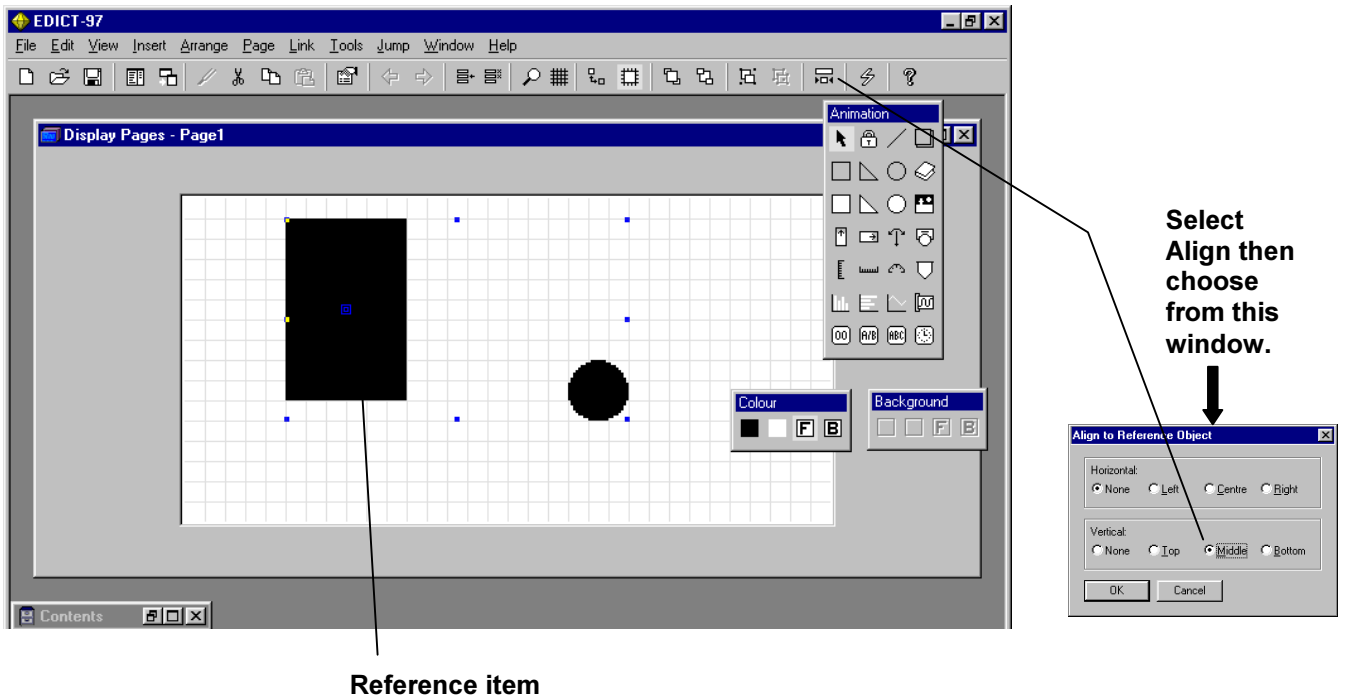


### Aligning animation items

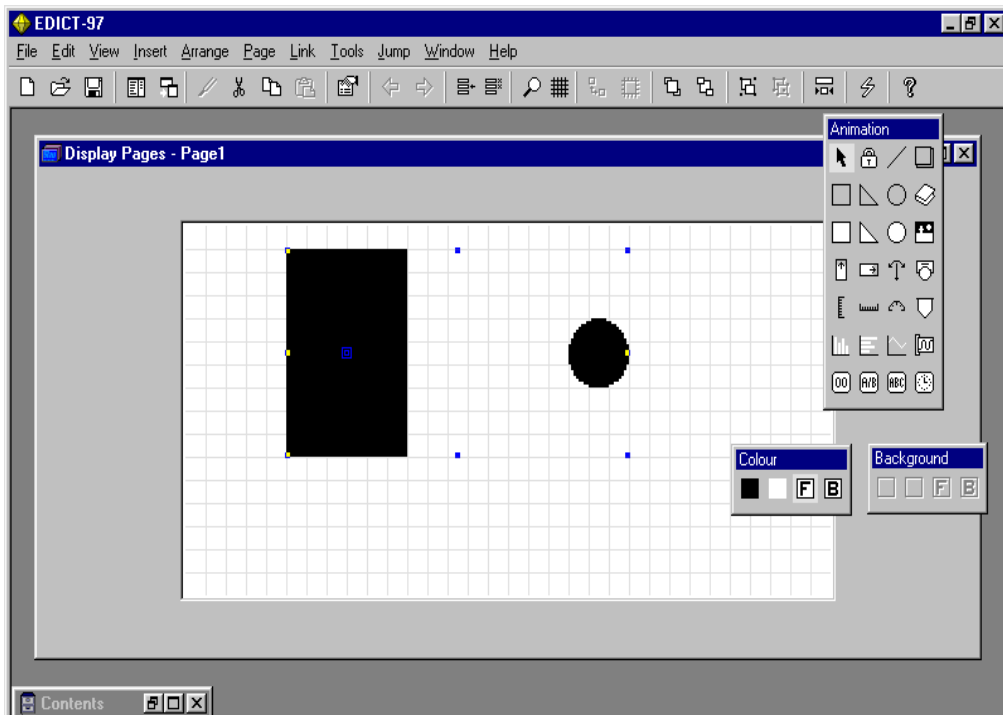
EDICT-97 provides facilities to align animation items with each other. To use this facility, first select the reference item to which the other items are to be aligned. Hold down the Shift key and then select the items that you wish to align. Note that the reference item will contain a small square to identify it as such. Once you have the items selected, select the "Align" command from the "Arrange" menu or select the Align Icon from the Toolbar at the top of the page. This will display a dialog box, which allows you to select how you want the items aligned.

The "Align" command can also be used to align a single object within the display page boundaries. Select the item, and then select the command. The item will be aligned relative to the display page. If you want to apply this function to multiple items, select them all, and then convert them into a group. Perform the alignment, and then ungroup the items to return them to their original status.

Aligning the rectangle and disk in our example.



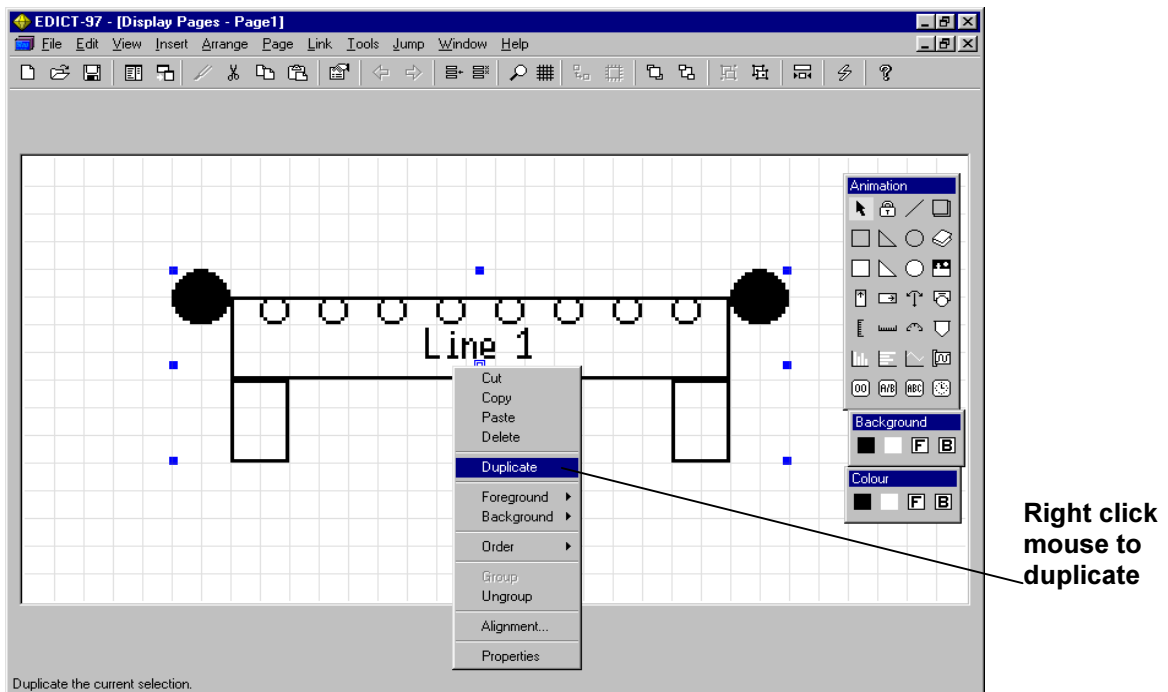
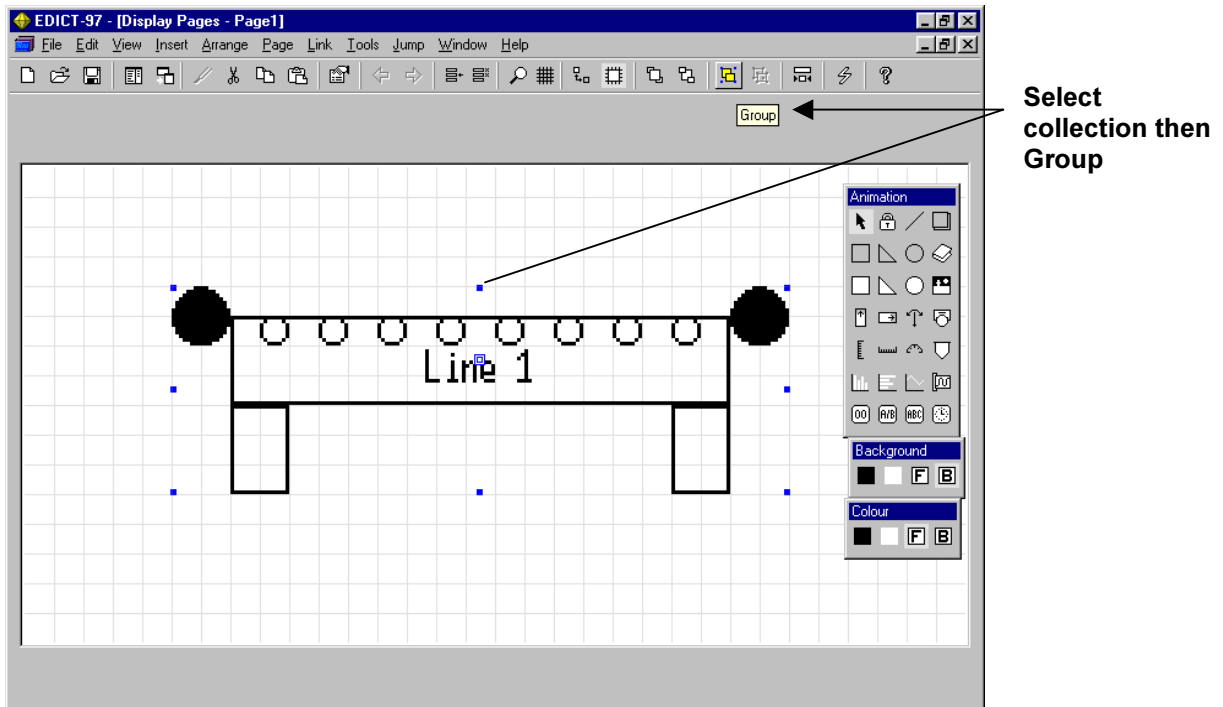
The disk is now aligned Vertical/Middle with the rectangle.

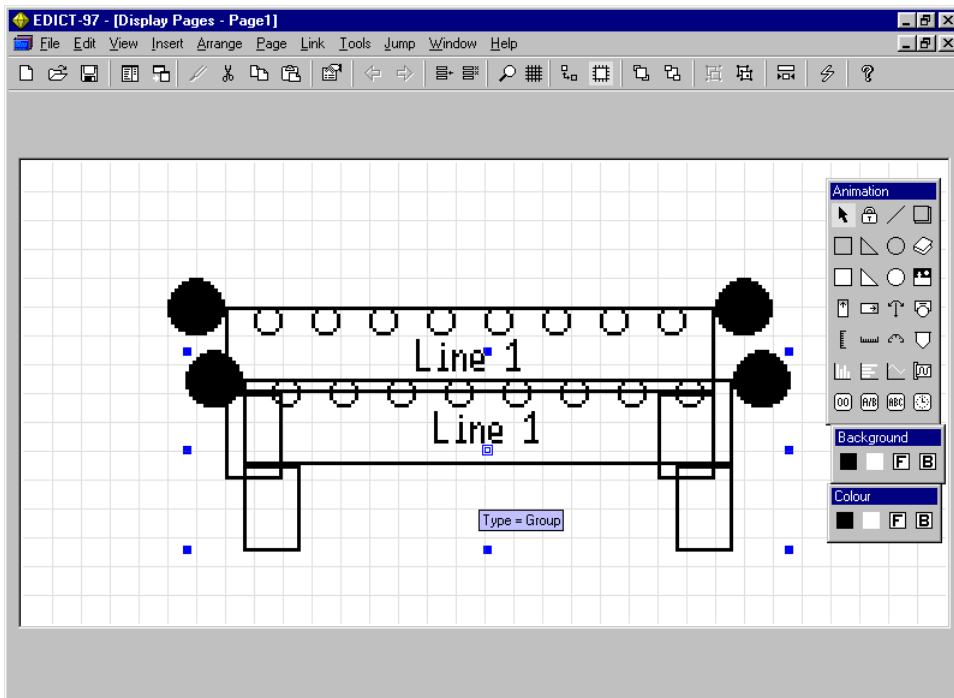


## Grouping animation items

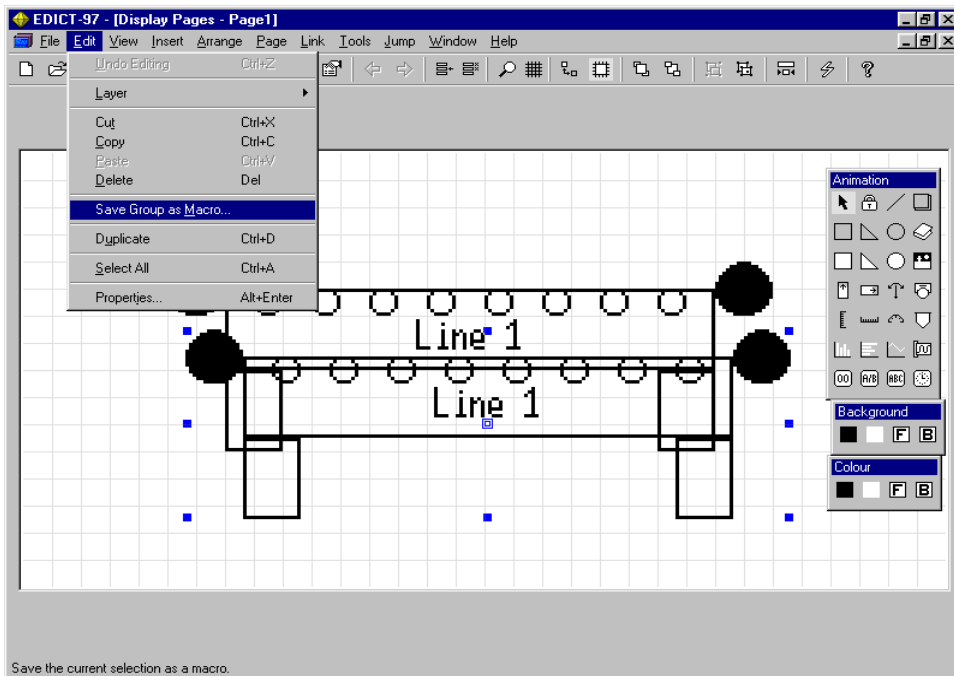
Animation items can be combined into a group. A group is a collection of items, which is manipulated as a single item. This can be useful when building library symbols, or when using the same collection of items many times over. Groups can be nested, such that a group can contain other groups up to any sensible level. To create a group, select the required items, and select the “Group” command from the menu. To split a group into its component items, select the group item, and then select the “Ungroup” command.

The following example shows how a collection of 3 rectangles, 8 circles, 2 disks and 1 line of Graphics layer fixed text can be combined into a **Group**.



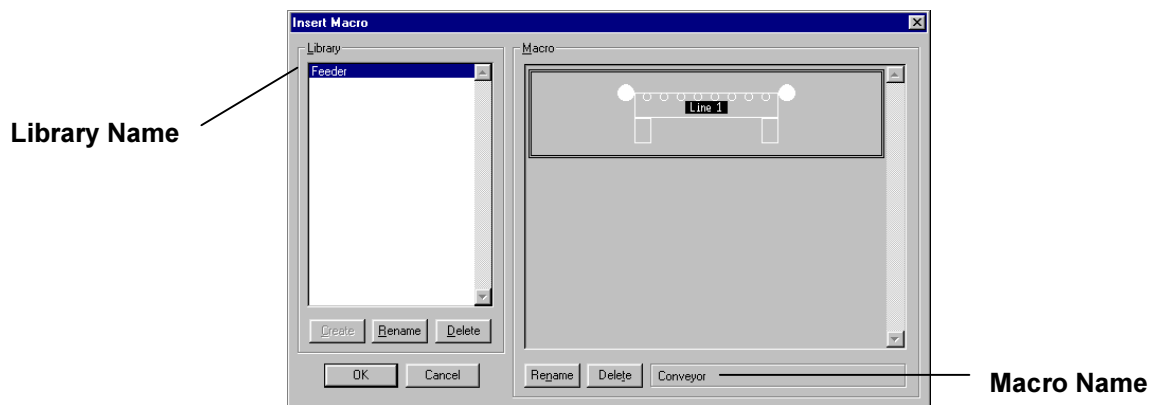
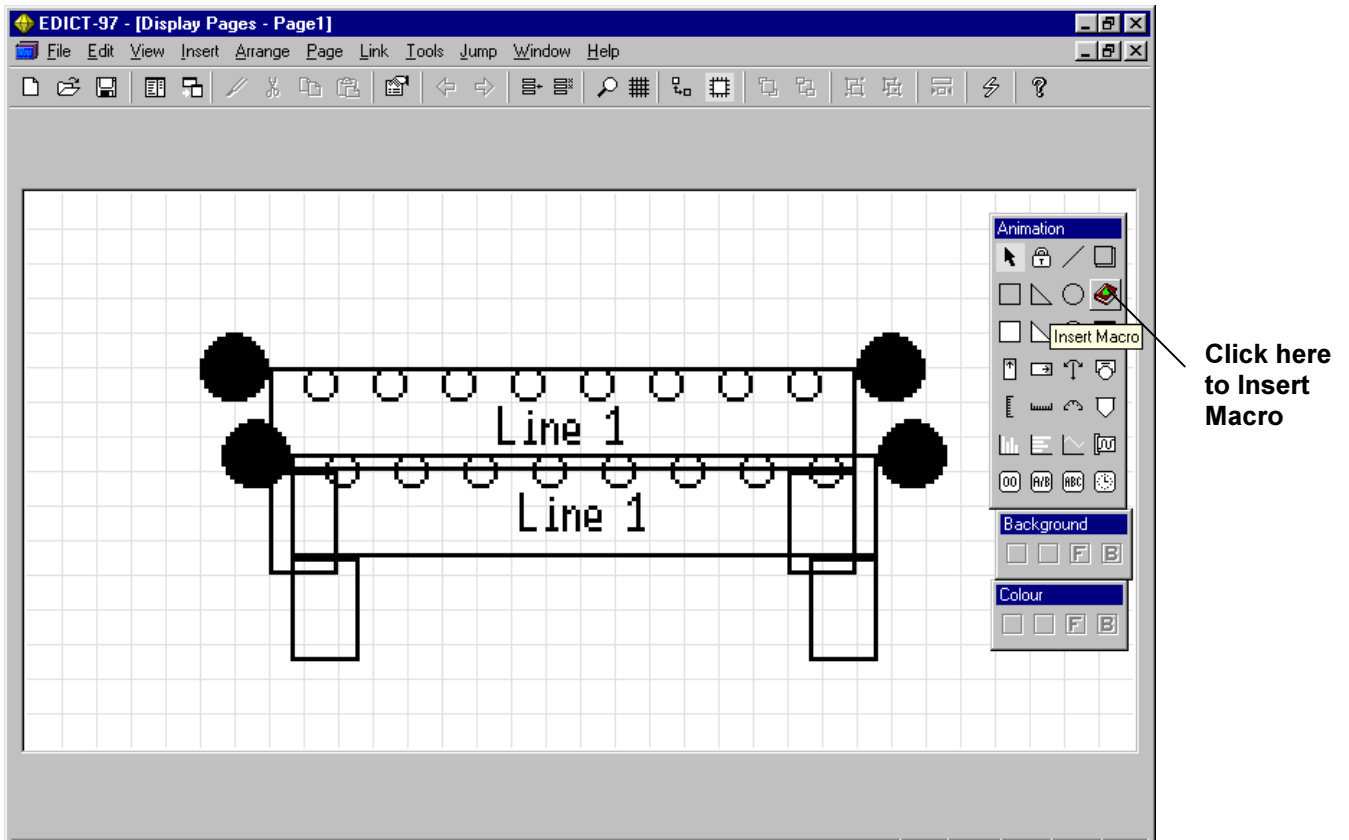


To save this **Group** as **Macro**



**Save Group as Macro**  
**Macro Library: Feeder**  
**Macro Name: Conveyor**

To Insert **Macro** from **Library**



## Using Group Properties in Macros

EDICT-97 has 5 different Group Properties, which provide the designer great flexibility when creating Macros. These Group Properties are **Group Color, Group Value, Group Data, Group String and Group Action**. These powerful features can be used on both monochrome and color graphic units.

**Group Color:** Each Macro can contain up to 8 different Group Colors. The color of individual or multiple animation items in a Macro may be defined as a Group Color (1 to 8). When a Group is saved as a Macro, the designer has the option to use an alias for this Group Color. For example a Macro for a grain hopper may be created where the color of a rectangle may be assigned the property GroupColor1. When this Macro is saved, the alias "Hopper Color" is substituted for the property GroupColor1. When the Macro is selected from the library, the property "Hopper Color" will appear in the properties window of this Macro.

**Group Value:** Each Macro can contain up to 8 different Group Value items. The value of individual or multiple animation items in a Macro may be defined as a Group Value item (1 to 8). Animation items designated as Group Value items must be "read only" variables. When a Group is saved as a Macro the designer has the option to use an alias for this Group Value item. For example a Macro for a grain hopper may be created where a Vertical Fill animation item is used to represent the level of grain in the hopper. The value of the Vertical Fill animation item is assigned the property GroupValue1. When this Macro is saved, the alias "Hopper Level" is substituted for the property GroupValue1. When the Macro is selected from the library, the property "Hopper Level" will appear in the properties window of this Macro.

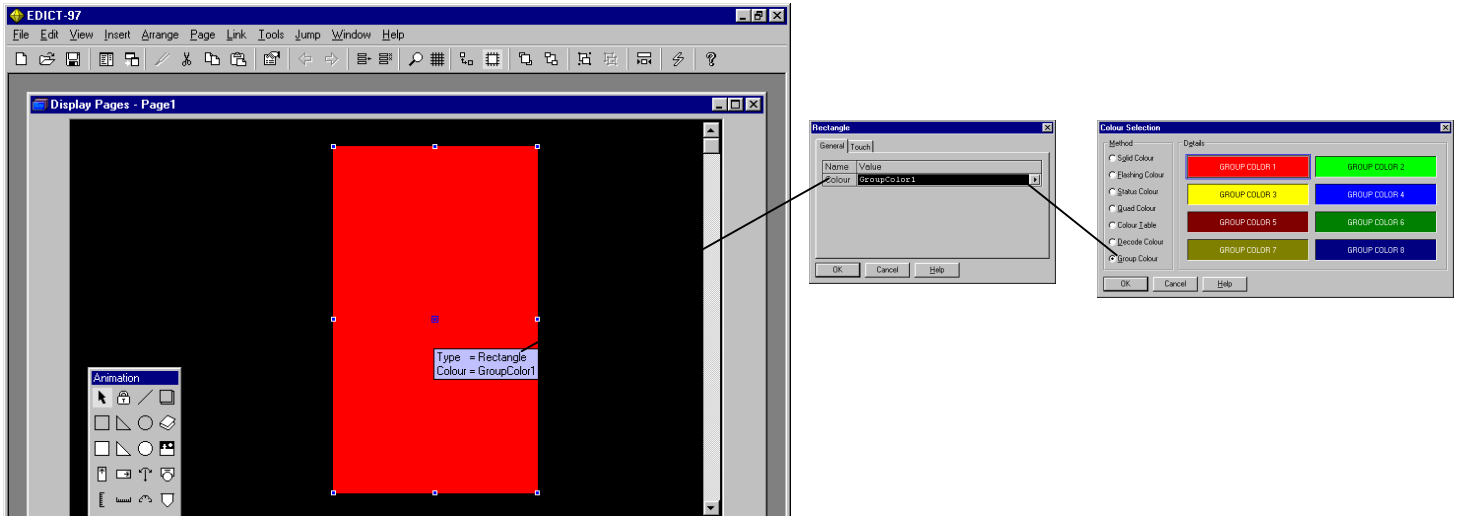
**Group Data:** Each Macro can contain up to 4 different Group Data items. The value of individual or multiple animation items in a Macro may be defined as a Group Data item (1 to 4). Group Data items are similar to Group Value items, but they can also be "data entry" variables. When a Group is saved as a Macro, the designer has the option to use an alias for this Group Data item. For example a Macro for a grain hopper may be created where a Insert Integer (Data entry) animation item is used to enter the amount of grain that is going to be dispensed from the hopper. The value of the Insert Integer is assigned the property GroupData1. When this Macro is saved, the alias "Dispense" is substituted for the property GroupData1. When the Macro is selected from the library, the property "Dispense" will appear in the properties window of this Macro.

**Group String:** Each Macro can contain up to 4 different Group String items. The value of individual or multiple animation items in a Macro may be defined as a Group String item (1 to 4). When a Group is saved as a Macro, the designer has the option to use an alias for this Group String item. For example a Macro for a grain hopper may be created where a General Text animation item is used to enter the hopper name. The value of the General Text item is assigned the property GroupString1. When this Macro is saved, the alias "Hopper Name" is substituted for the property GroupString1. When the Macro is selected from the library, the property "Hopper Name" will appear in the properties window of this Macro.

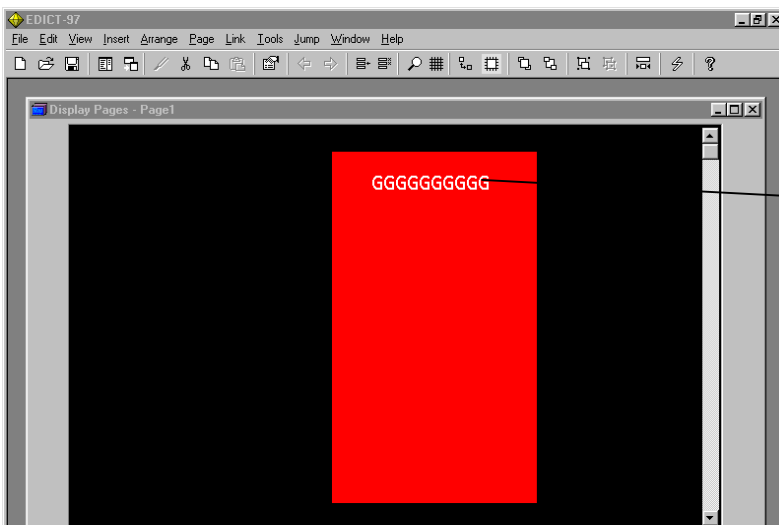
**Group Action:** Each Macro can contain up to 4 different Group Action items. The value of individual or multiple animation items in a Macro may be defined as a Group Action item (1 to 4). When a Group is saved as a Macro, the designer has the option to use an alias for this Group Action item. For example, a Macro for a grain hopper may be created where a disk animation item is used to start the dispensing process. The disk is designated "touch sensitive" and the "On Pressed" property is assigned GroupAction1. When this Macro is saved, the alias "Start Bit" is substituted for the property GroupAction1. When the Macro is selected from the library, the property "Start Bit" will appear in the properties window of this Macro.

## Creating the Macro for the grain hopper

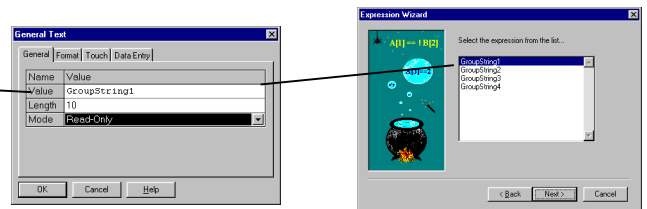
Insert rectangle and set the rectangle's color as GroupColor1



Insert General Text and set the General Text's value to GroupString1

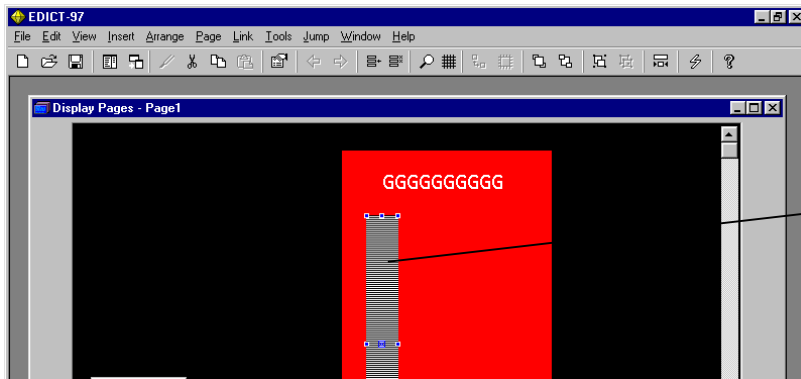


Using the  
Expression Wizard (Variable/System)

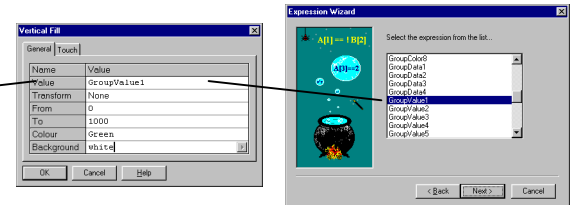


Note: The background color of the General Text and Inserted Integer items are set to GroupColor1. The background of the General Text and Inserted Integer items will match the color of the rectangle.

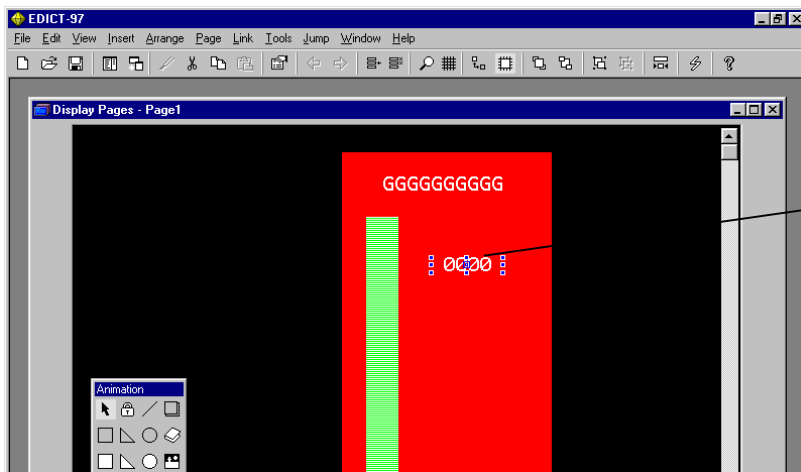
Insert Vertical Fill and set the value property to GroupValue1.



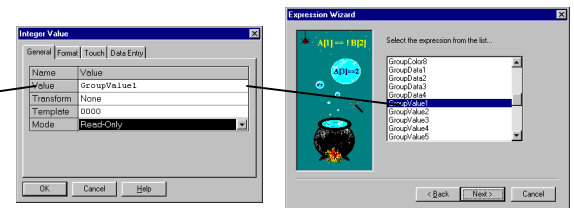
Using the  
Expression Wizard (Variable/System)



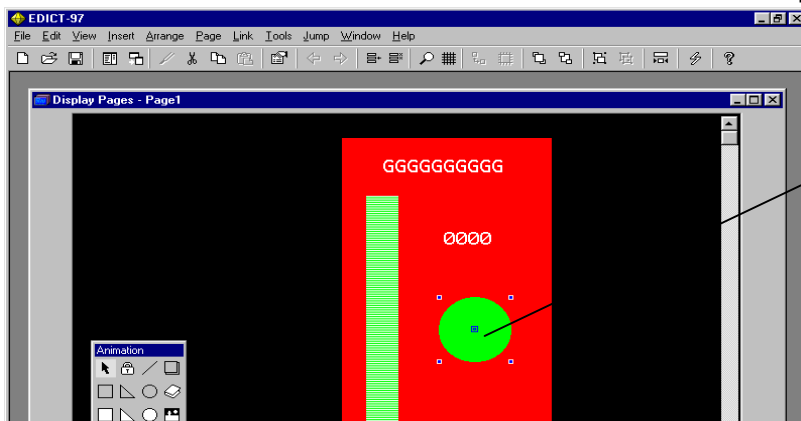
Insert Integer Value and set the value property to GroupValue1



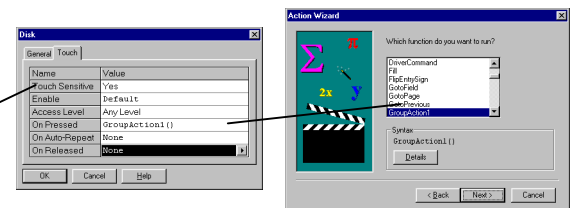
Using the  
Expression Wizard (Variable/System)



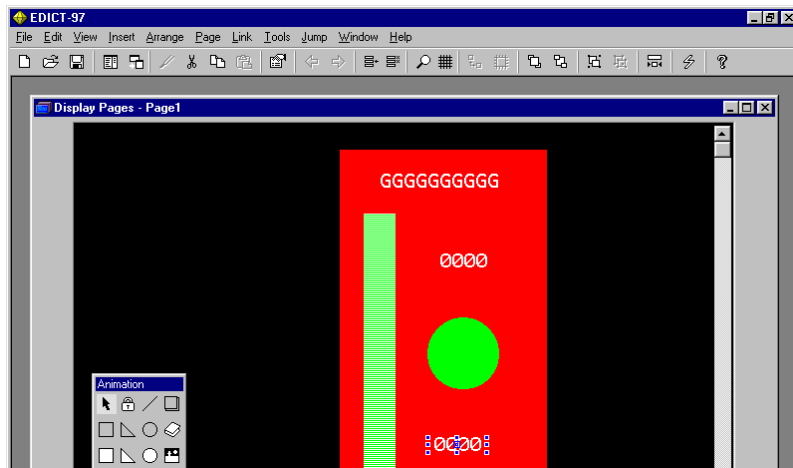
Insert Disk Animation item and set "On Pressed to GroupAction1"



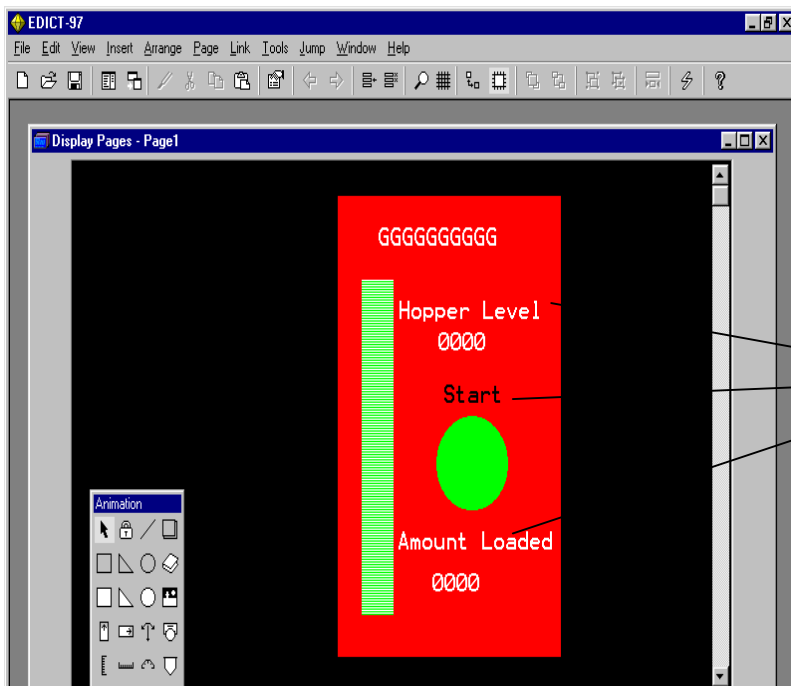
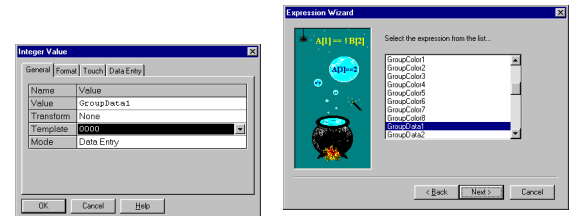
Using the  
Action Wizard  
(Variable/Neither of the above)



**Insert Integer Value (data entry) and set value property to GroupData1**



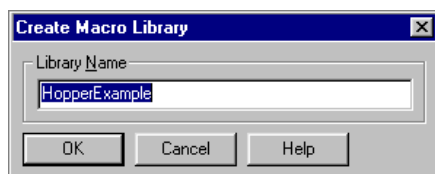
**Using the Expression Wizard (Variable/System)**



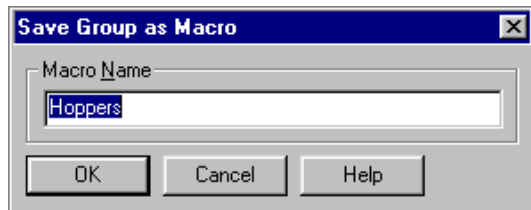
**Inserted Fixed Text**

**Now Group all of the animation items and save as Macro**

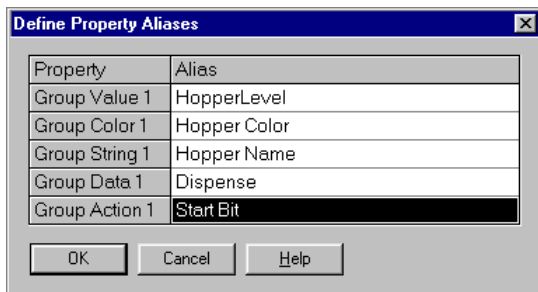
The following windows will appear.



**Create the Macro Library**

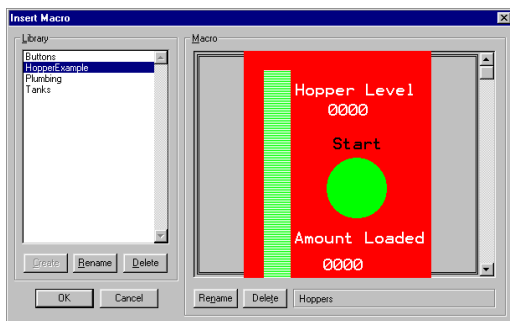


← Name the Macro

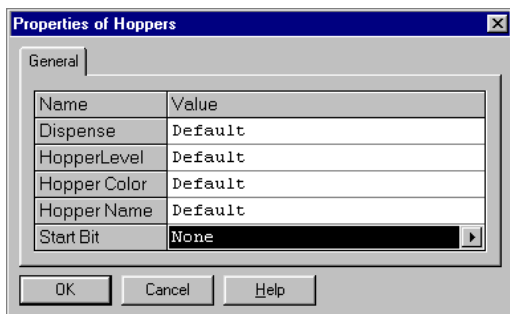


← Define the Property Aliases for the Macro

On a display page, insert the Macro Hoppers from the HopperExample library



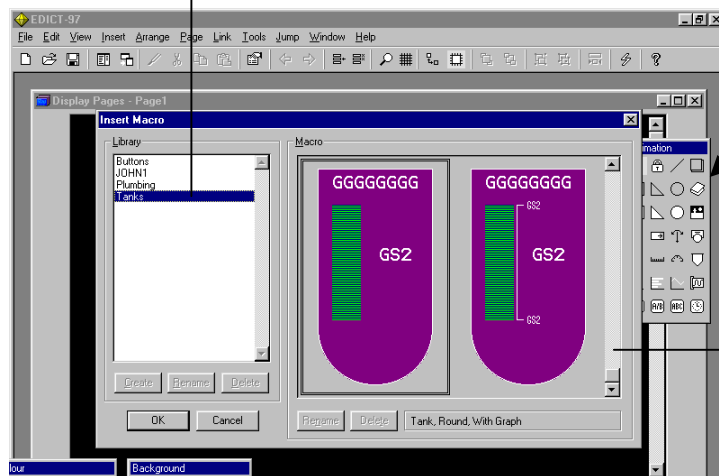
Double click on the inserted Hoppers Macro and the following window appears



Enter the names of the values for this Macro  
For example:  
Dispense=HopLoad  
HopperLevel=LevelSensor1  
Hopper Color=Navy  
Hopper Name=Wheat  
Start Bit=HopStart

## Inserting animation items from EDICT-97's library

### 2) Select Library

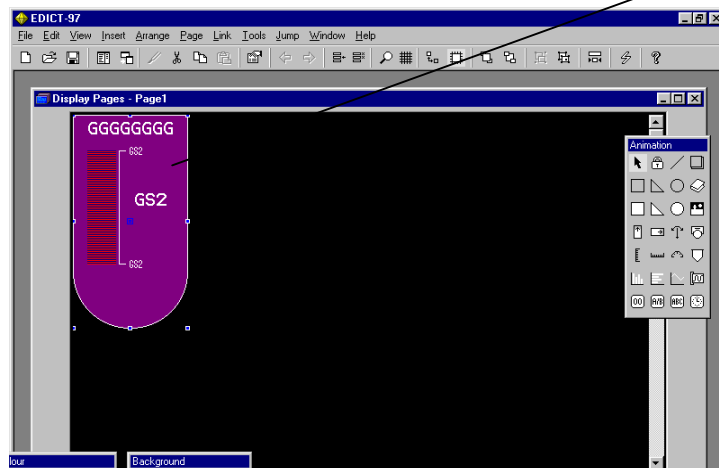


1) Select Insert from Macro Library

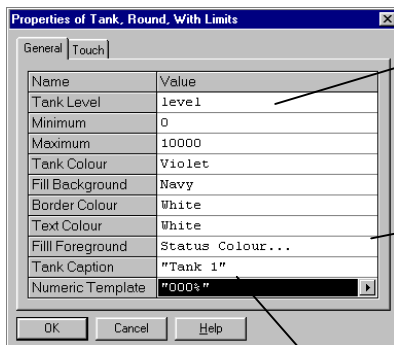
3) Use scroll bar to make selection

4) Insert on page

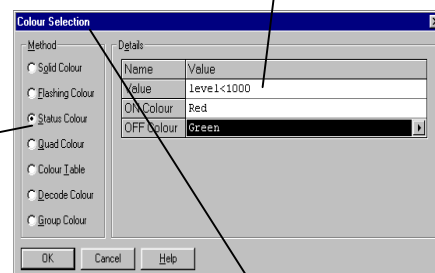
5) Double click on item to edit properties



Fill color will change from green to red when the value "level" falls below 1000



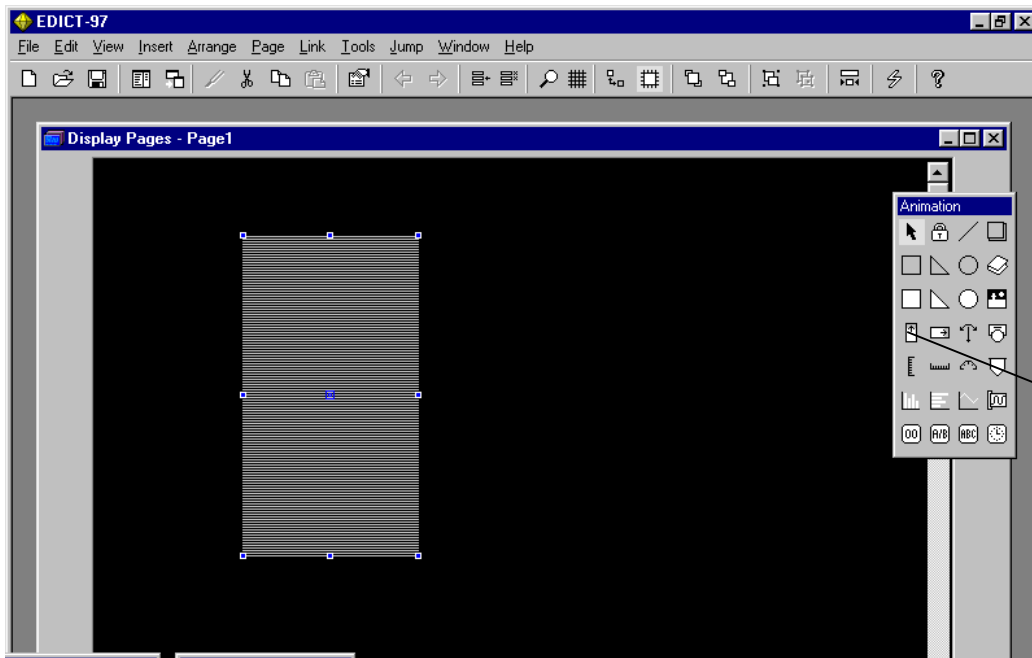
level reads the value of a PLC register



To edit caption properties use F2 key and arrow keys. Type in desired caption (make sure text is within " ")

See Page E41 for details on color selection options

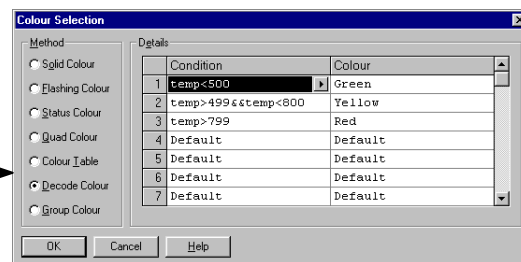
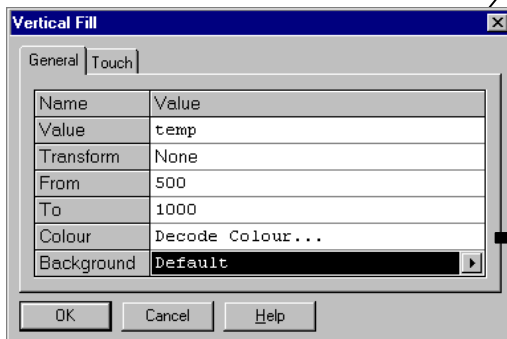
## Inserting a Horizontal or Vertical Fill



Insert Vertical Fill  
onto a display  
page

Double click on  
inserted Vertical  
Fill to edit  
properties

temp is reading the  
value of a Red Lion  
Controls temperature  
controller



The height of the Vertical Fill reflects the temperature of the temperature controller. When the temperature is under 500 F the color of the fill is green. Between 500 F and 800 F the fill is yellow. The color of the fill will be red at temperatures of 800 F or higher.

### The Horizontal Fill Animation Item

This animation item draws a horizontal bar, which varies in size based upon the controlling value. You can define the foreground and background colors to be used, plus an optional transformation for the controlling value.

The table below lists the properties of this animation item...

Property	Description
Value	An integer expression defining the value to be displayed.
Transform	An optional transform to be performed on the data. Before the data value is displayed, it will be transformed using the information in this property. This can be used, for example, to scale values from PLC to engineering units. Note that the "Value" property is always evaluated as a 16-bit number, and that transforms that manipulate bits will thus behave as if they have been passed a 16-bit argument, even if the underlying data is really a 32-bit value.
From	An integer expression defining the minimum value to display. Data values less than this value are clipped before being displayed. This value can be numerically greater than the "To" property if you want the graph to work "backwards".
To	An integer expression defining the maximum value to display. Data values greater than this value are clipped before being displayed. This value can be numerically less than the "From" property if you want the graph to work "backwards".
Color	The Color to be used for the bar. See Color Selection Table for options (Page E41)
Background	The Color to be used for the section of the rectangle not taken up by the bar. See Color Selection Table for options (Page E41)

### The Vertical Fill Animation Item

This animation item draws a vertical bar, which varies in size based upon the controlling value. You can define the foreground and background Colors to be used, plus an optional transformation for the controlling value.

The table below lists the properties of this animation item...

Property	Description
Value	An integer expression defining the value to be displayed.
Transform	An optional transform to be performed on the data. Before the data value is displayed, it will be transformed using the information in this property. This can be used, for example, to scale values from PLC to engineering units. Note that the "Value" property is always evaluated as a 16-bit number, and that transforms that manipulate bits will thus behave as if they have been passed a 16-bit argument, even if the underlying data is really a 32-bit value.
From	An integer expression defining the minimum value to display. Data values less than this value are clipped before being displayed. This value can be numerically greater than the "To" property if you want the graph to work "backwards".
To	An integer expression defining the maximum value to display. Data values greater than this value are clipped before being displayed. This value can be numerically less than the "From" property if you want the graph to work "backwards".
Color	The Color to be used for the bar. See Color Selection Table for options (Page E41)
Background	The Color to be used for the section of the rectangle not taken up by the bar.

### The Horizontal Scale Animation Item

This animation item displays a horizontal “comb” which can be used to label an axis or a horizontal fill. The comb is made up of a number of major divisions; each of which can be further subdivided into a number of minor divisions.

The table below lists the properties of this animation item...

Property	Description
Major Divisions	The number of major divisions on the scale. This represents the number of gaps to be shown by the comb, as opposed to the number of ticks. The number of ticks will be one more than the number of gaps.
Minor Divisions	The number of minor divisions per major division. If you enter a value of 1 or None, the minor divisions will not be shown. Again, this value represents the number of gaps and not the number of ticks, although this time the number of ticks will be one less than the number of gaps.
Color	The Color to be used for the comb. See Color Selection Table for options (Page E41).
Orientation	The orientation of the comb. This option controls which direction the tick marks on the comb point. You may combine two combs with opposite orientation with a fill effect to create an attractive meter.

### The Vertical Scale Animation Item

This animation item displays a vertical “comb” which can be used to label an axis or a vertical fill. The comb is made up of a number of major divisions; each of which can be further subdivided into a number of minor divisions.

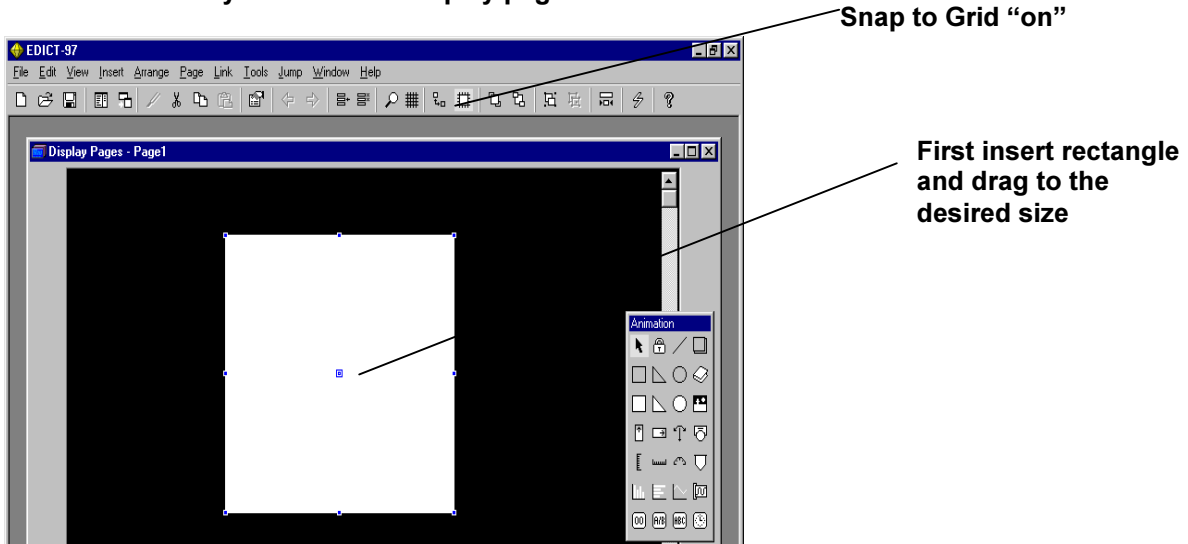
The table below lists the properties of this animation item...

Property	Description
Major Divisions	The number of major divisions on the scale. This represents the number of gaps to be shown by the comb, as opposed to the number of ticks. The number of ticks will be one more than the number of gaps.
Minor Divisions	The number of minor divisions per major division. If you enter a value of 1 or None, the minor divisions will not be shown. Again, this value represents the number of gaps and not the number of ticks, although this time the number of ticks will be one less than the number of gaps.
Color	The Color to be used for the comb. See Color Selection Table for options (Page E41)
Orientation	The orientation of the comb. This option controls which direction the tick marks on the comb point. You may combine two combs with opposite orientation with a fill effect to create an attractive meter.

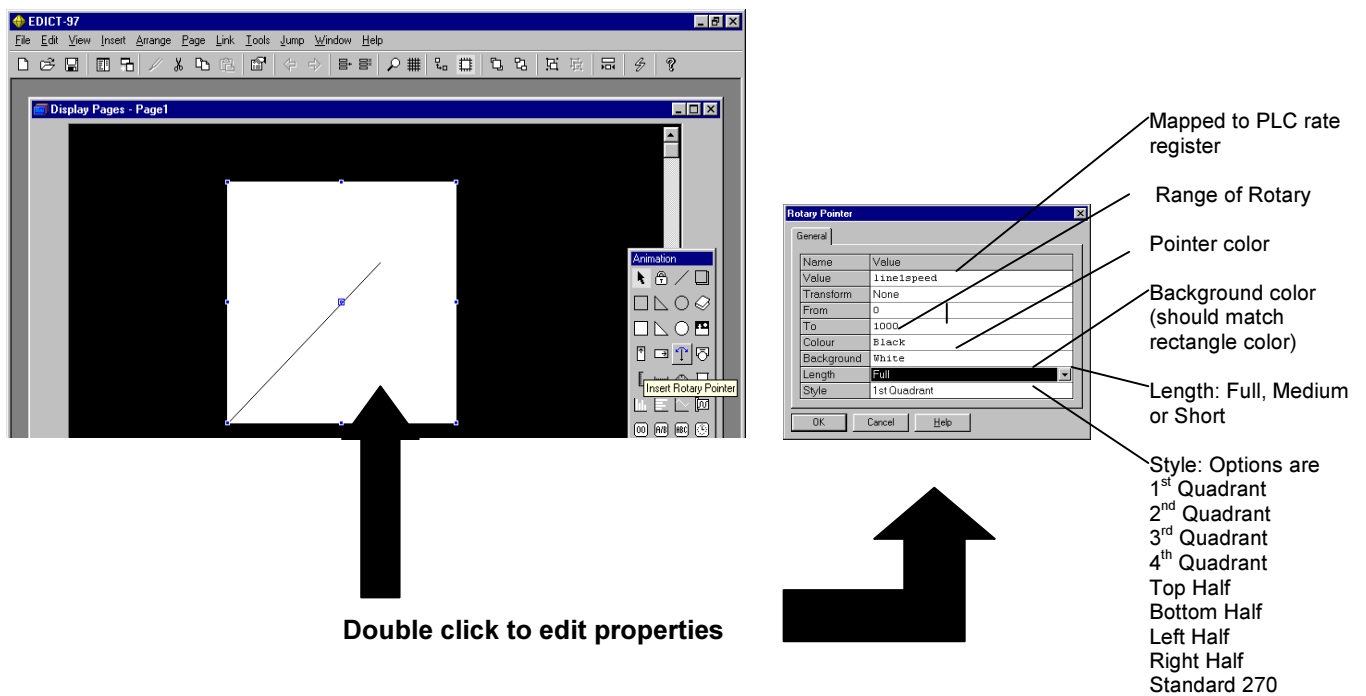
## Inserting a Rotary Pointer

EDICT-97 has 9 different styles of Rotary Pointers along with 9 different Rotary scales. The Rotary scales have major and minor divisions settings that provide the designer great flexibility for creating custom animation items.

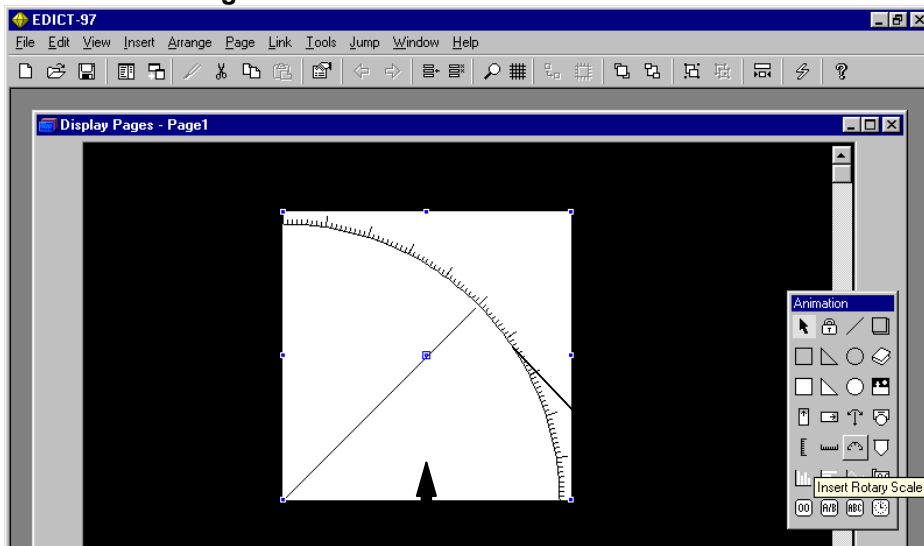
### To Insert a Rotary Pointer on a display page



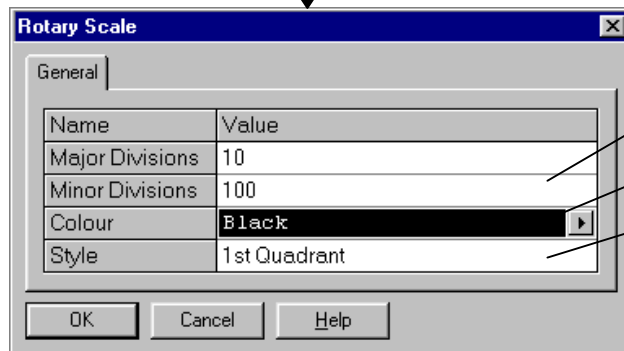
Insert Rotary Pointer and drag it (by holding the left mouse button pressed) to match the size of the rectangle



**Insert Rotary Scale and drag it (by holding the left mouse button pressed) to match the size of the rectangle**



**Double click to edit properties**



**Set Major and Minor divisions of Scale**

Color of Scale

Style should match style of Rotary Pointer

### The Rotary Pointer Animation Item

This animation item draws a rotary pointer, which varies in position based upon the controlling value. You can define the foreground and background colors to be used, plus an optional transformation for the controlling value.

The table below lists the properties of this animation item...

Property	Description
Value	An integer expression defining the value to be displayed.
Transform	An optional transform to be performed on the data. Before the data value is displayed, it will be transformed using the information in this property. This can be used, for example, to scale values from PLC to engineering units. Note that the "Value" property is always evaluated as a 16-bit number, and that transforms that manipulate bits will thus behave as if they have been passed a 16-bit argument, even if the underlying data is really a 32-bit value.
From	An integer expression defining the minimum value to display. Data values less than this value are clipped before being displayed. This value can be numerically greater than the "To" property if you want the pointer to work "backwards".
To	An integer expression defining the maximum value to display. Data values greater than this value are clipped before being displayed. This value can be numerically less than the "From" property if you want the pointer to work "backwards".
Color	The Color to be used for the rotary pointer. See Color Selection Table for options (Page E41)
Background	The Color to be used for pointer background area. This should be selected to match the color of the area where the pointer is placed.  For Example if the pointer is placed on a blue rectangle the background color of the pointer should be blue.
Length	Three options: Full, Medium or Short
Style	Nine options: 1 <sup>st</sup> Quadrant, 2 <sup>nd</sup> Quadrant, 3 <sup>rd</sup> Quadrant, 4 <sup>th</sup> Quadrant, Top Half, Bottom Half, Left half, Right Half and Standard 270 degree

**The Rotary Scale Animation Item**

This animation item displays a rotary “comb” which can be used to label a rotary pointer. The comb is made up of a number of major divisions; each of which can be further subdivided into a number of minor divisions.

**The table below lists the properties of this animation item...**

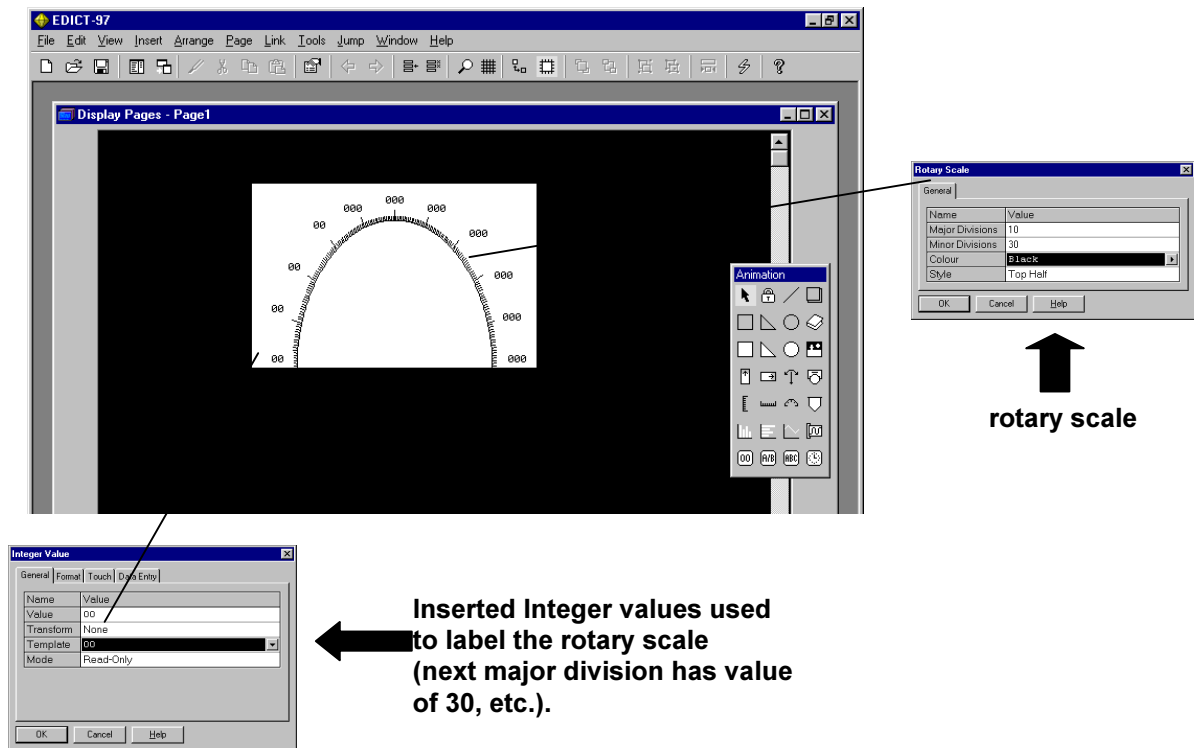
Property	Description
Major Divisions	The number of major divisions on the scale. This represents the number of gaps to be shown by the comb, as opposed to the number of ticks. The number of ticks will be one more than the number of gaps.
Minor Divisions	The number of minor divisions per major division. If you enter a value of 1 or None, the minor divisions will not be shown. Again, this value represents the number of gaps and not the number of ticks, although this time the number of ticks will be one less than the number of gaps.
Color	The Color to be used for the comb. See Color Selection Table for options (Page E41).
Style	The style of the comb. There are nine different styles to match the nine styles of rotary pointers.

## Using Multiple Rotary pointers

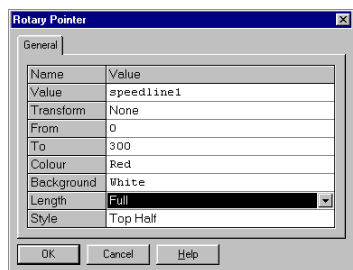
Multiple rotary pointers may be placed on the same area. This allows designers to create custom multiple variable rotary indicators. Setpoint vs. Actual value, relative position and rate comparisons are some examples where this feature can be used.

**This Example shows three line speeds displayed on one area.**

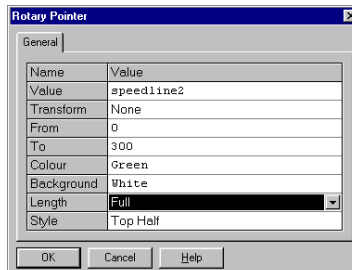
**Insert a rectangle and then rotary scale on a display page.**



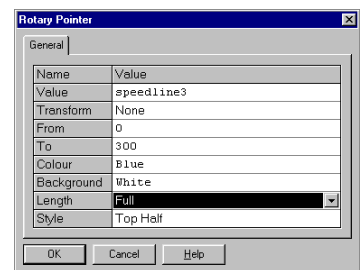
**Insert the three rotary pointers on the rectangle/rotary scale.**



**speedline1 is reading the rate value on a Red Lion Controls rate meter. This pointer is red.**



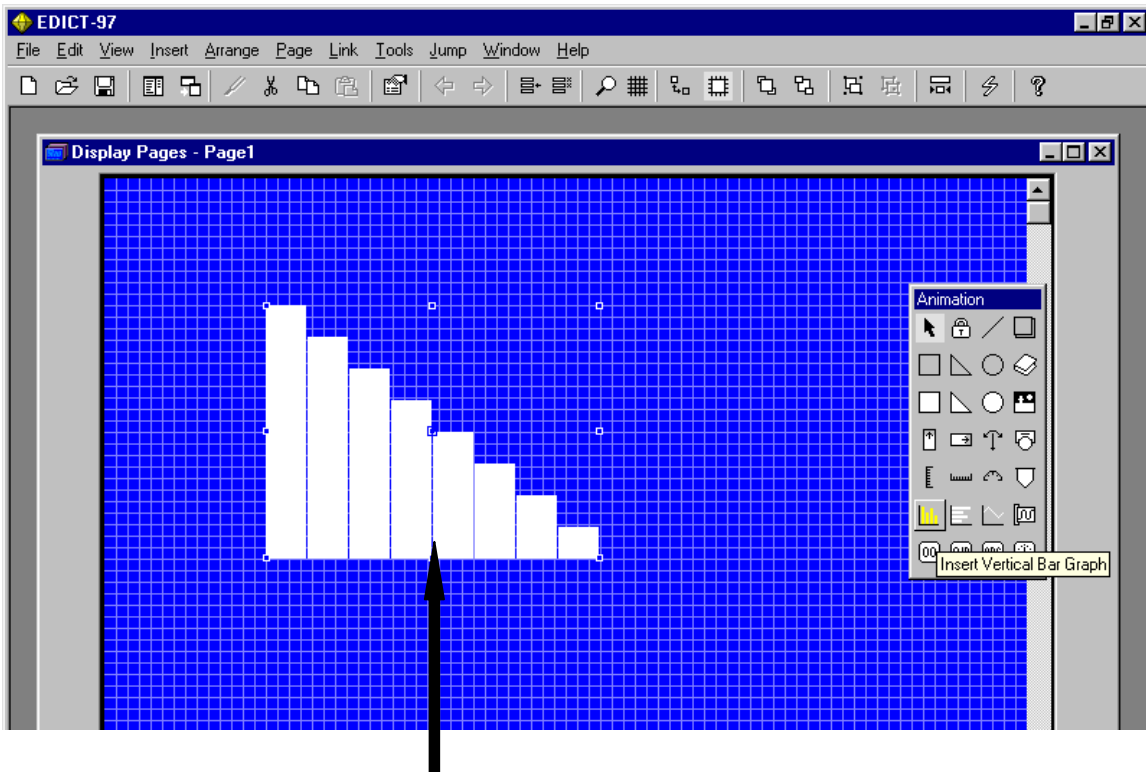
**speedline2 is reading the rate value on a Red Lion Controls rate meter. This pointer is green.**



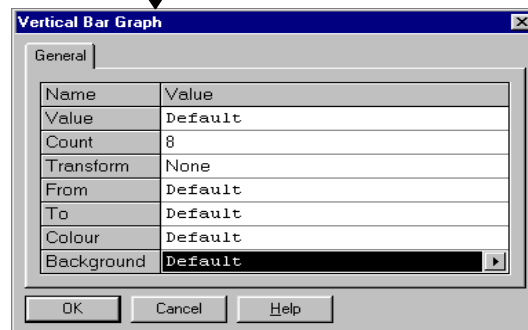
**speedline3 is reading the rate value on a Red Lion Controls Rate meter. This pointer is blue.**

## Inserting Bar Graph (Horizontal or Vertical) animation items on a page

Insert a Vertical Bar Graph from the animation tool box and drag it to the desired size



Double click to edit properties



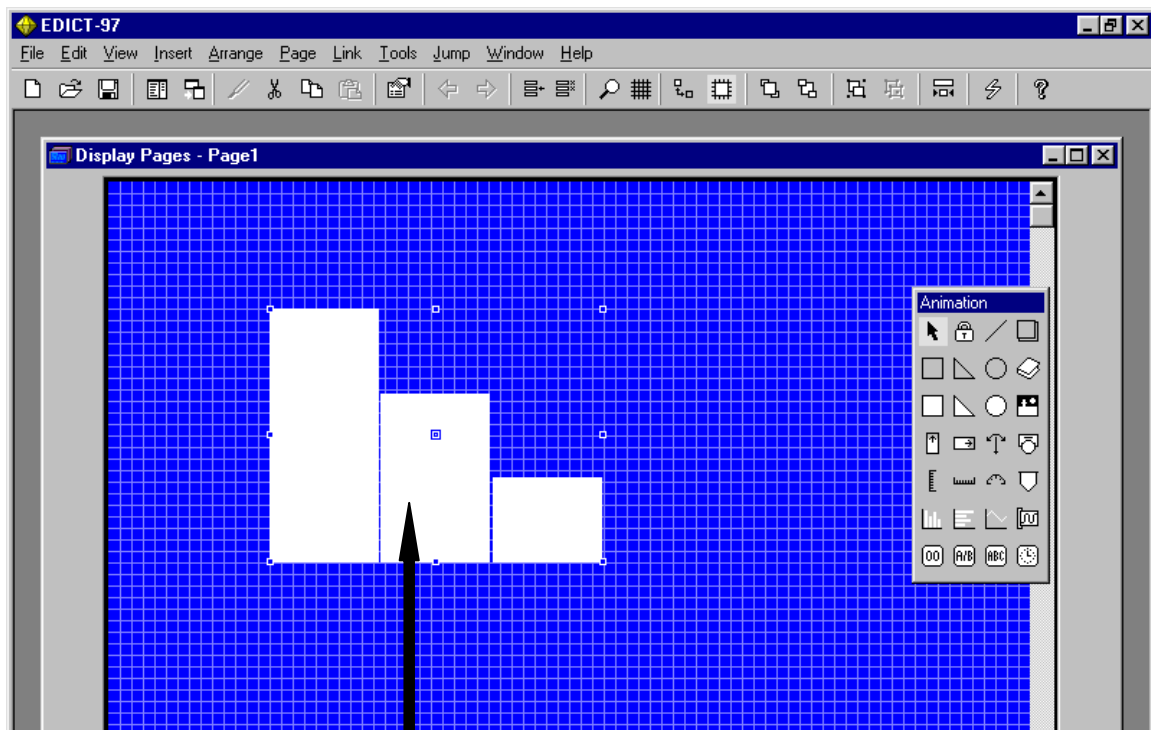
### The Vertical Bar Graph Animation Item

This animation item draws a number of vertical bars, which vary in size based upon values extracted from an array. You can define the foreground and background colors to be used, plus an optional transformation to be applied to each data value.

The table below lists the properties of this animation item...

Property	Description
Value	An indexable expression defining the values to be displayed. Values can be either CommsBlock values eg. A[0], A[1], A[2], etc. or arrays from the data files in Named Data e.g. pressure[0], pressure[1], pressure[2], etc.
Count	The number of data values to be displayed.
Transform	An optional transform to be performed on the data. Before each data value is displayed, it will be transformed using the information in this property. This can be used, for example, to scale values from PLC to engineering units. Note that the "Value" property is always evaluated as a 16-bit number, and that transforms that manipulate bits will thus behave as if they have been passed a 16-bit argument, even if the underlying data is really a 32-bit value.
From	An integer expression defining the minimum value to display. Data values less than this value are clipped before being displayed. This value can be numerically greater than the "To" property if you want the graph to work "backwards".
To	An integer expression defining the maximum value to display. Data values greater than this value are clipped before being displayed. This value can be numerically less than the "From" property if you want the graph to work "backwards".
Color	The Color to be used for the bars. See Color Selection Table for options (Page E41).
Background	The Color to be used for the section of the rectangle not taken up by the bars. See Color Selection Table for options (Page E41).

## Using values from CommsBlocks



Double click to edit properties

**Vertical Bar Graph**

General

Name	Value
Value	A[0]
Count	3
Transform	None
From	0
To	100
Colour	Default
Background	Default

OK Cancel Help

Value set to A[0] designates that A[0] is the starting point of the CommsBlock array. Count of 3 indicates that there are 3 values in the array. They are A[0], A[1] and A[2]. Each of the Vertical bars will have values ranging from 0 to 100.

**Named Data**

Name	Maps To	Data Type	Transform
1 yeast	A[0]	16-bit Signed	None
2 hops	A[1]	16-bit Signed	None
3 barley	A[2]	16-bit Signed	None

A[0], A[1] and A[2] are mapped to Data Registers in a GE Fanuc Series 90 PLC.

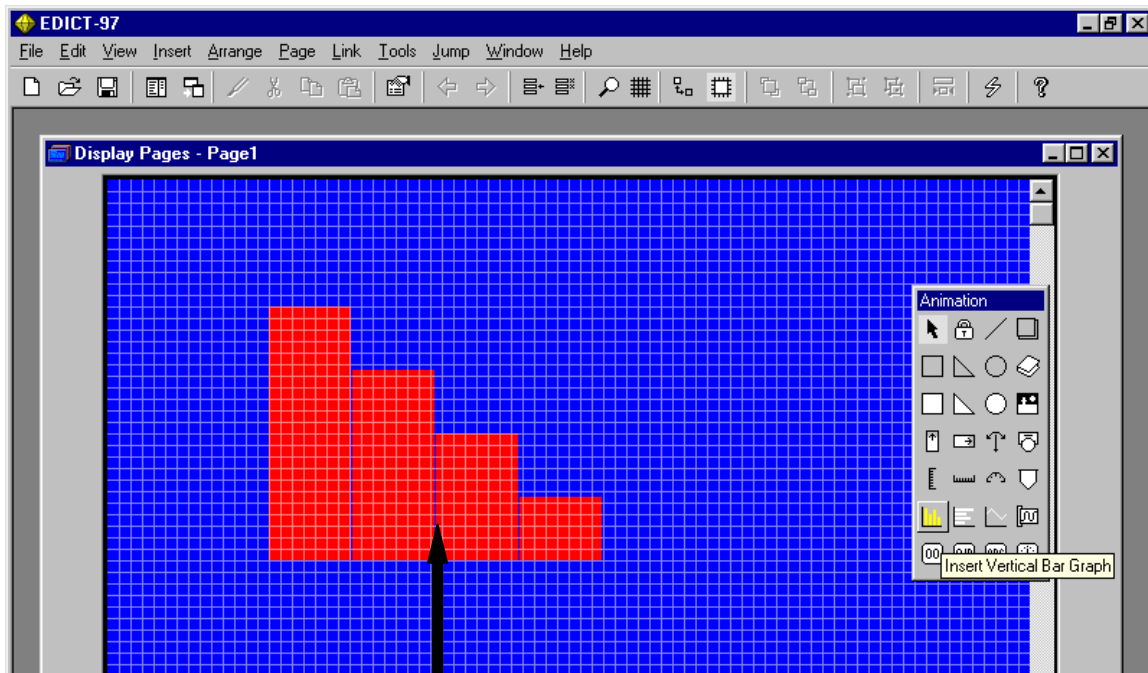
In this example NamedData was used to give A[0], A[1] and A[2] the variable names yeast, hops and barley.

**Block Address List**

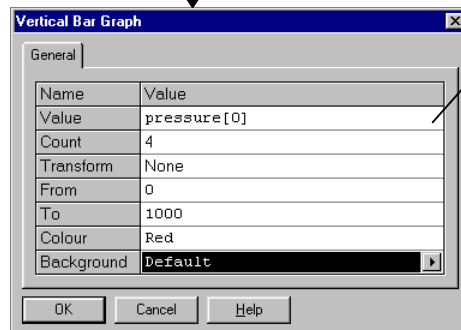
Register	Address	Comment
A[0]	%R0001	None
A[1]	%R0002	None
A[2]	%R0003	None

OK Cancel Help

## Using values from Data Files in Named Data



Double click to edit properties



Value set to pressure[0] designates that pressure[0] is the starting point for the Data Files array. Count of 4 indicates that there are 4 values in this array. They are pressure[0], pressure[1], pressure[2] and pressure[3]. The values of each bar will range from 0 to 1000. The color of each bar will be red.

Name	Maps To	Size	Data Type
1 pressure	Internal	4	16-bit Signed
2 None	Internal	Auto	16-bit Signed

This example uses the indirect addressing feature of EDICT-97 to log data variables. In this example, the operator interface is reading the value of a Red Lion Controls IAMS smart signal conditioner.

### The Horizontal Bar Graph Animation Item

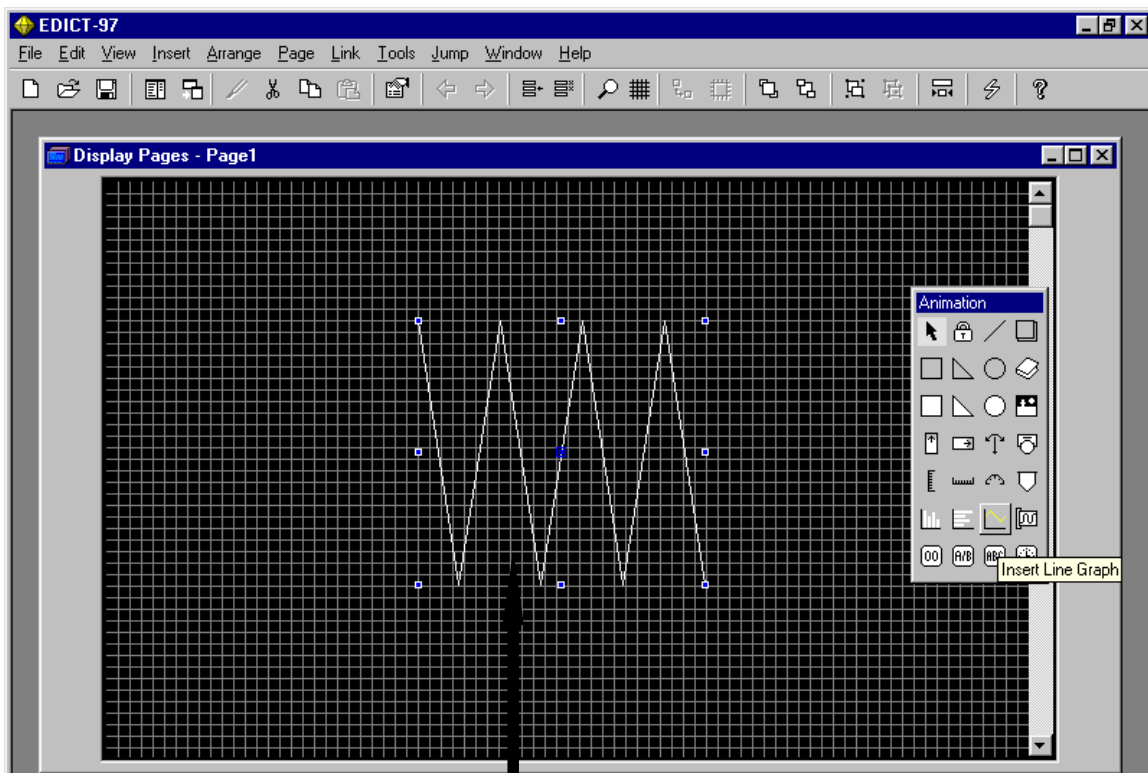
This animation item draws a number of horizontal bars, which vary in size based upon values extracted from an array. You can define the foreground and background colors to be used, plus an optional transformation to be applied to each data value.

The table below lists the properties of this animation item...

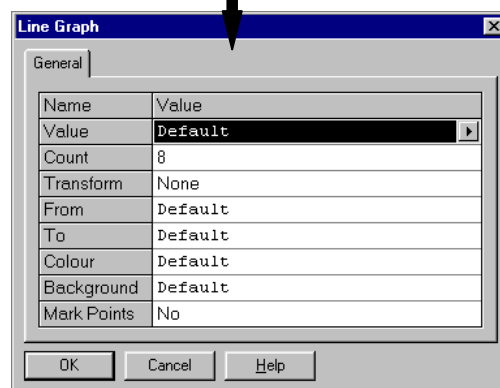
Property	Description
Value	An indexable expression defining the values to be displayed. Values can be either CommsBlock values, eg. A[0], A[1], A[2], etc., or arrays from the data files
Count	The number of data values to be displayed.
Transform	An optional transform to be performed on the data. Before each data value is displayed, it will be transformed using the information in this property. This can be used, for example, to scale values from PLC to engineering units. Note that the "Value" property is always evaluated as a 16-bit number, and that transforms that manipulate bits will thus behave as if they have been passed a 16-bit argument, even if the underlying data is really a 32-bit value.
From	An integer expression defining the minimum value to display. Data values less than this value are clipped before being displayed. This value can be numerically greater than the "To" property if you want the graph to work "backwards".
To	An integer expression defining the maximum value to display. Data values greater than this value are clipped before being displayed. This value can be numerically less than the "From" property if you want the graph to work "backwards".
Color	The Color to be used for the bars. See Color Selection Table for options (Page E41).
Background	The Color to be used for the section of the rectangle not taken up by the bars. See Color Selection Table for options (Page E41).

## Inserting Line Graph Animation items on a display page

Insert a Line Graph from the animation tool box and drag to desired size



Double click to edit properties



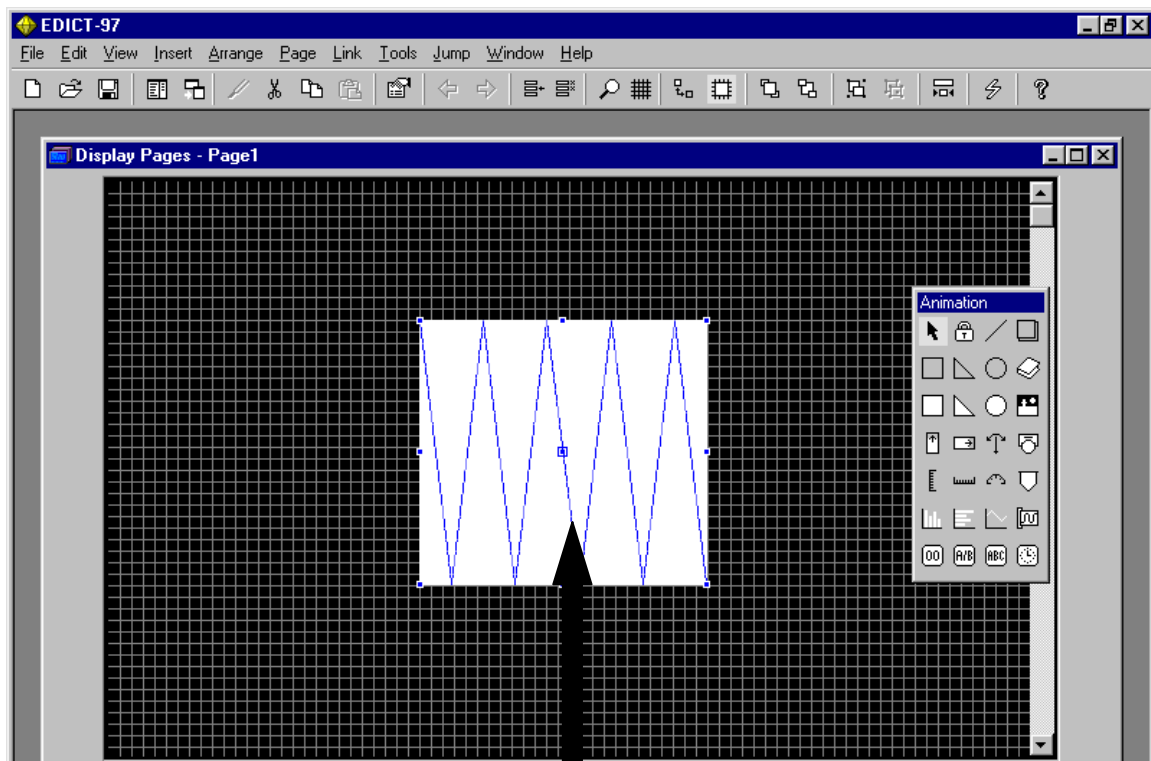
### The Line Graph Animation Item

This animation item draws a line graph based upon data values extracted from an array. You can define the foreground and background colors to be used, plus an optional transformation to be applied to each data value.

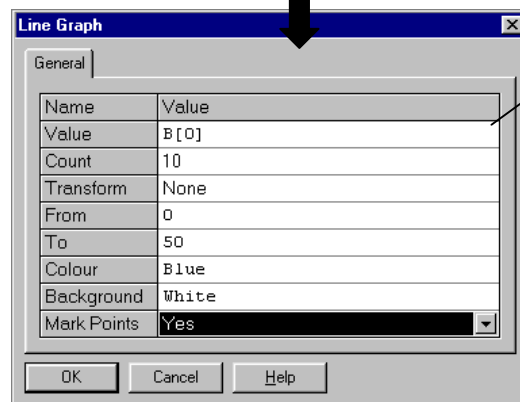
The table below lists the properties of this animation item...

Property	Description
Value	An indexable expression defining the values to be displayed. Values can be either CommsBlock values eg. A[0], A[1], A[2], etc. or arrays from the data files in Named Data e.g. pressure[0], pressure[1], pressure[2], etc.
Count	The number of data values to be displayed.
Transform	An optional transform to be performed on the data. Before each data value is displayed, it will be transformed using the information in this property. This can be used, for example, to scale values from PLC to engineering units. Note that the "Value" property is always evaluated as a 16-bit number, and that transforms that manipulate bits will thus behave as if they have been passed a 16-bit argument, even if the underlying data is really a 32-bit value.
From	An integer expression defining the minimum value to display. Data values less than this value are clipped before being displayed. This value can be numerically greater than the "To" property if you want the graph to work "backwards".
To	An integer expression defining the maximum value to display. Data values greater than this value are clipped before being displayed. This value can be numerically less than the "From" property if you want the graph to work "backwards".
Color	The Color to be used for the line and data point markers. See Color Selection Table for options (Page E41).
Background	The Color to be used for the section of the rectangle not taken up by the graph. Either enter an expression, which evaluates to one of the Color values, or select a Color from the list displayed by the Alt+Down key combination.
Mark Points	Whether or not to mark the data points with small rectangles.

## Using values from CommsBlocks



Double click to edit properties

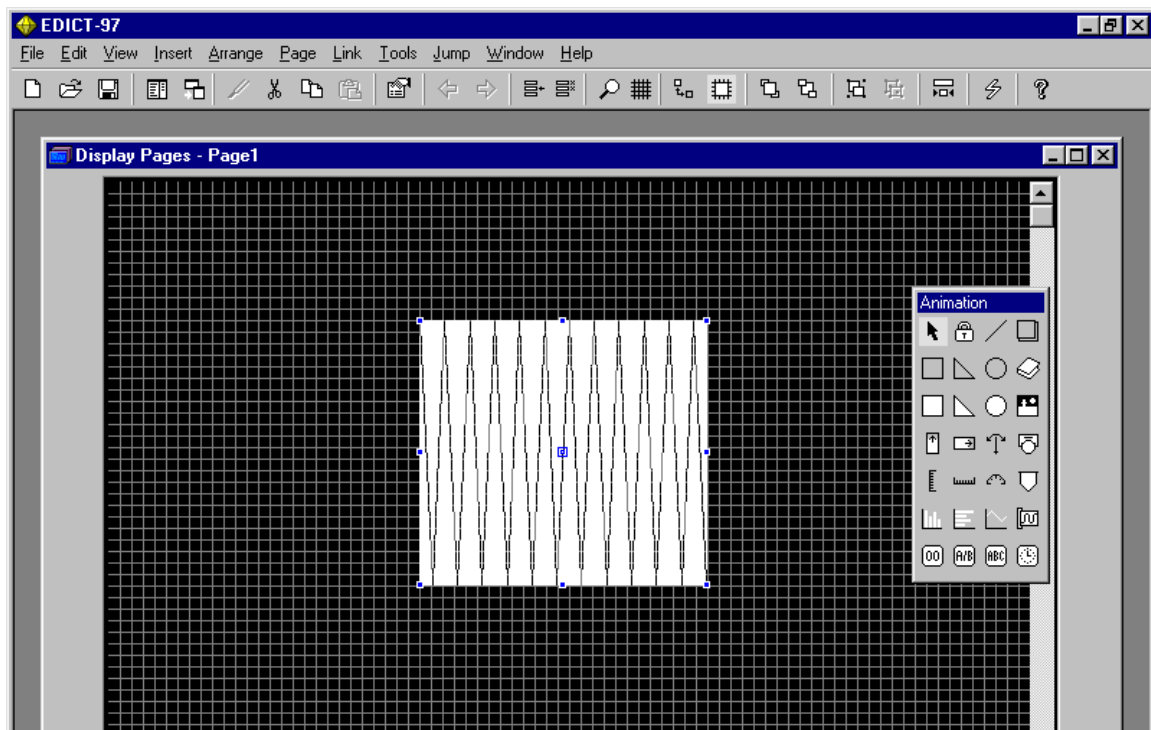


Value set to B[0] designates that B[0] is the starting point of the CommsBlock array. Count of 10 indicates that there are 10 values in the array. They are B[0], B[1]... B[9]. Each of the points on the line will have values ranging from 0 to 50 on the Y axis. The color of the line and points on the line is blue. The background color is white.

Internal CommsBlock reference

Device	Address	Data Type	Size	Access	Update
A	None	None	0	Read	Auto
B	Internal	16-bit Signed	10	Read	Auto
C	None	16-bit Signed	0	Read	Auto
D	None	None	0	Read	Auto

## Using values from Data Files in Named Data



**Line Graph**

General

Name	Value
Value	Drums[0]
Count	24
Transform	None
From	0
To	5000
Colour	Black
Background	White
Mark Points	Yes

OK Cancel Help

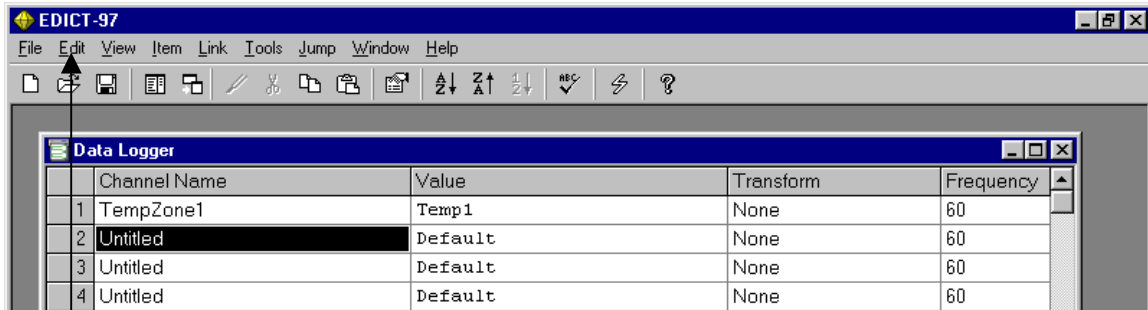
A chemical manufacturer wants to plot hourly production. Triggered by the Schedule table the operator interface reads a PLC counter register every hour for hourly production. The operator interface stores these values in the Named Data array Drums. The count is 24 for 24 hours (one reading/hour). Hourly production ranges from 0 to 5000 drums/hour.

## Using the Data Logger

The Data Logger can be used for the source of displayed trends. Trends can be displayed on a preformatted display page ( by choosing **Page Properties/Category/TrendViewer**) or by using the **Insert Trend** animation item from the Graphics Layer Animation ToolBox.

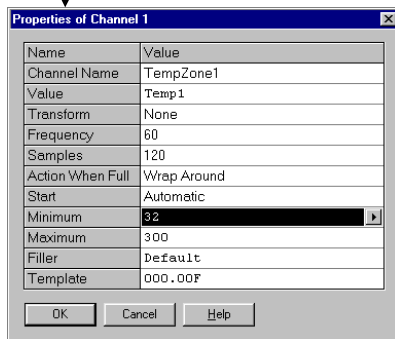
## Using the Trend Viewer on a Display Page

(The following Channel in the Data Logger was used as the source for the Trend)



	Channel Name	Value	Transform	Frequency
1	TempZone1	Temp1	None	60
2	Untitled	Default	None	60
3	Untitled	Default	None	60
4	Untitled	Default	None	60

Highlight TempZone 1 and choose Edit properties.

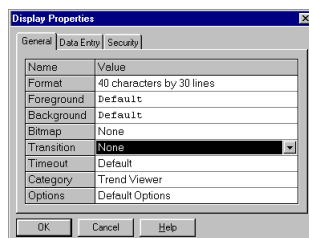
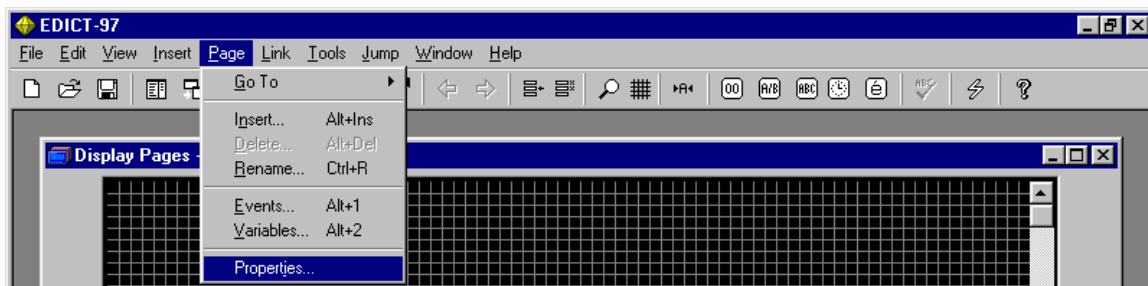


Name	Value
Channel Name	TempZone1
Value	Temp1
Transform	None
Frequency	60
Samples	120
Action When Full	Wrap Around
Start	Automatic
Minimum	32
Maximum	300
Filler	Default
Template	000.00F

OK Cancel Help

The source for TempZone 1 (Channel 1) is the value Temp1. Temp1 reads the process value from a Red Lion Controls PID temperature controller. A frequency of 60 indicates that the operator interface reads this process value once a minute (60 seconds). This channel will log 120 readings (samples) and wrap around when the log is full (newest readings will replace oldest readings when log is full). The logged value will range from 32 to 300 F).

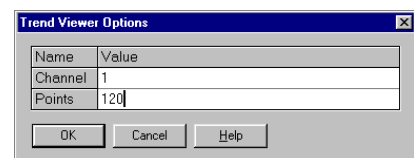
## Using the preformatted trend viewer display page (Select Page/Properties)



Name	Value
Format	40 characters by 30 lines
Foreground	default
Background	default
Bitmap	None
Transition	None
Timeout	Default
Category	Trend Viewer
Options	Default Options

OK Cancel Help

Select Category **Trend Viewer** and select Channel 1 from options. Set **Points** to 120. The Trend will display all 120 values in TempZone1.

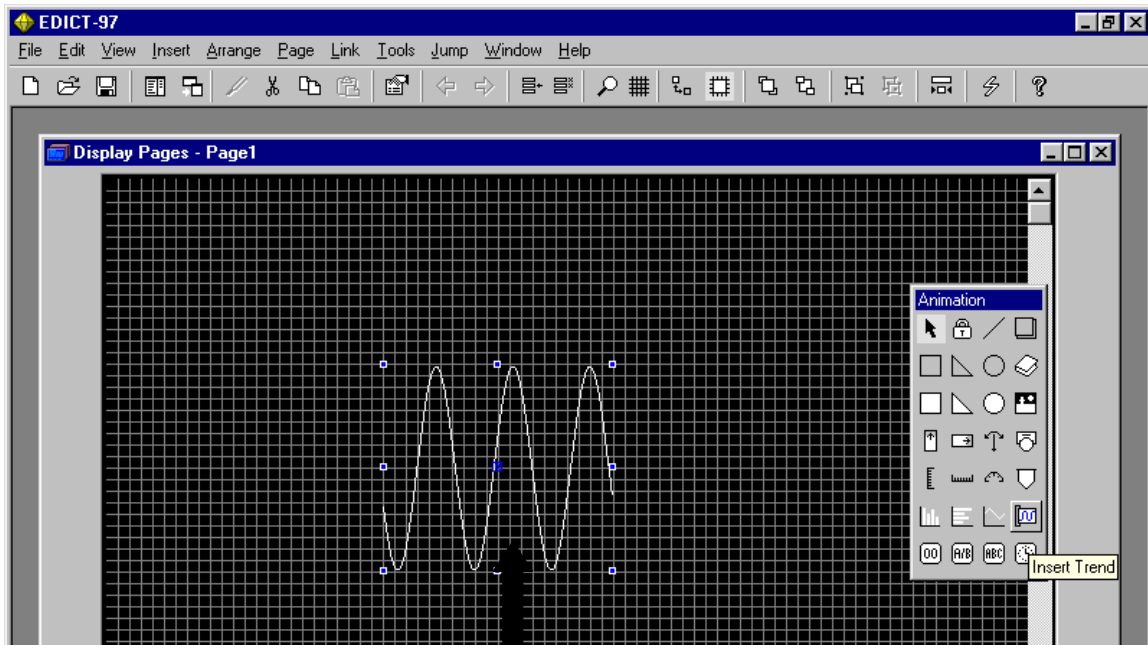


Name	Value
Channel	1
Points	120

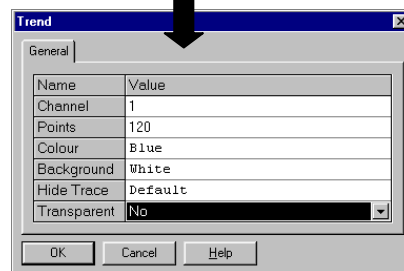
OK Cancel Help

## Displaying a Trend on the Graphics Layer of a Display Page

Insert the View Trend animation item from the graphics layer animation toolbox and drag to desired size



Double click to edit properties



Channel 1 is TempZone1 from the Data Logger. All 120 logged values will be displayed. The background color is white and the trend line color is blue. The Trace is not hidden. A condition or expression could be used to hide a Trace. For example, if Temp1>100 was entered into the **Hide Trace** window, the Trend Line would not be visible when Temp1 exceeded 100F. Transparent refers to multiple trends displayed in the same area.

Using the same example to display multiple trends on one area (multiple pens)

**Note:** The number of trends that may be displayed is limited only by the number of colors (16 solid colors and 240 flashing combinations). Multiple trends must have the same time base (frequency).

From Channel 2 in the Data Logger

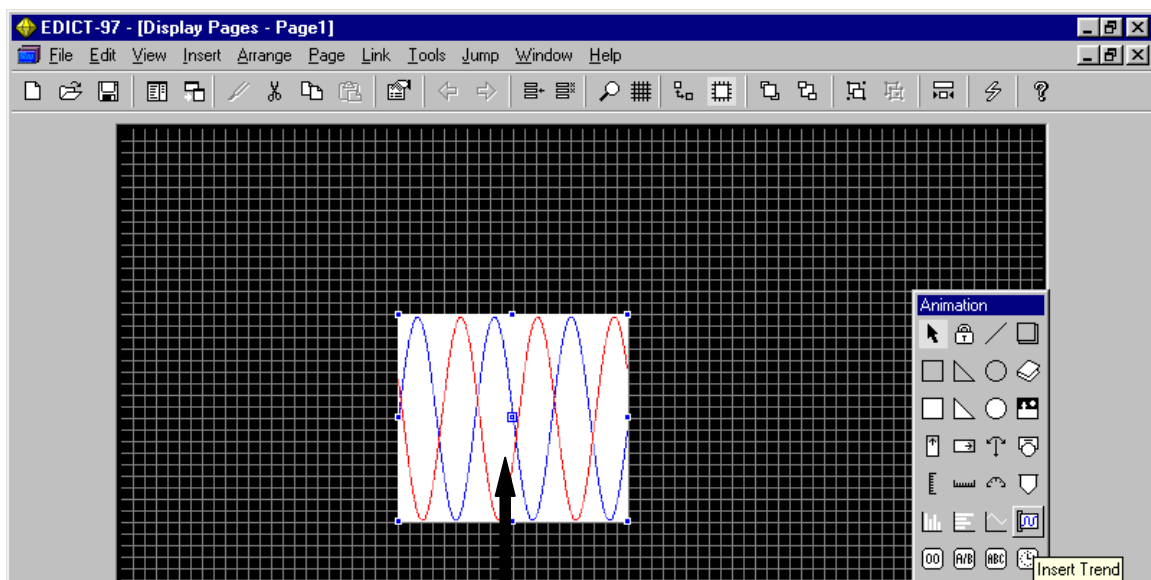
Properties of Channel 2

Name	Value
Channel Name	TempZone1SP
Value	Zone1SP
Transform	None
Frequency	1
Samples	120
Action When Full	Default
Start	Automatic
Minimum	32
Maximum	300
Filler	Default
Template	00000

OK Cancel Help

A second channel is configured to log the set point from the same Red Lion Controls PID temperature controller. The general properties match channel 1(TempZone1).

**Insert the second trend on top of the first trend**



**Double click to edit properties**

Trend

General

Name	Value
Channel	2
Points	120
Colour	Red
Background	White
Hide Trace	HideSP1==1
Transparent	Yes

OK Cancel Help

Channel 2 (value =Zone1SP) Trend is inserted on top of Channel 1 (value=Temp1). The line color of this trend is red. The background matches the background of the Channel 1 trend. By making this trend transparent, the viewer can see this trend along with the trend it covers. A variable called "HideSP1" was created and used to hide the trace when activated.

Event Map

Event	Enable	Action	Routing
1 Soft-key 1 pressed	Default	HideSP1:=!HideSP1	Default
2 None	Default	None	Default
3 None	Default	None	Default
4 None	Default	None	Default
5 None	Default	None	Default

**Soft key1 pressed toggles HideSP1.**  
(Hiding/UnHiding the Zone1SP Trend)

## The Color Selection Table

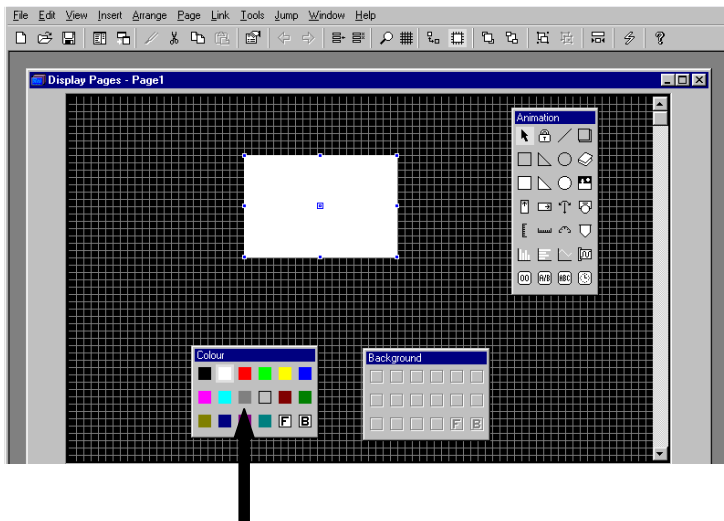
Graphical objects within EDICT-97 have a number of properties, which define their appearance. Some of these properties will define color. For example, a rectangle will have a single property, which defines the color in which the figure will be drawn. Similarly, a text item will have two properties, which define the item's foreground and background colors. In common with other EDICT-97 properties, color properties can be left undefined by the user. In this case, each property will assume a default value, which is appropriate to the property's application.

EDICT-97 allows default background and foreground colors to be defined either at the page or the database level. When a color property for an animation item is left in its default state, it will reflect either the default foreground or background color as appropriate. The majority of properties reflect the ambient foreground color, with only those properties, which specifically refer to the background of an item chosen to reflect the ambient background color. Example of the latter properties would be background properties of horizontal and vertical fill, or the background property of a text field.

If a color is not left at its default value, the user has several options as to how it can be defined. The simplest way is to select a single color. This can be done via the two color toolboxes, via the context menu for the animation item, or via the first page of the color selection dialog box. A further option allows the selection of a flashing color, whereby the user selects two colors, which will be displayed alternately at a rate defined elsewhere in the database. EDICT-97 offers all flashing combinations of its basic sixteen colors, resulting in a combined total 256 solid and flashing colors.

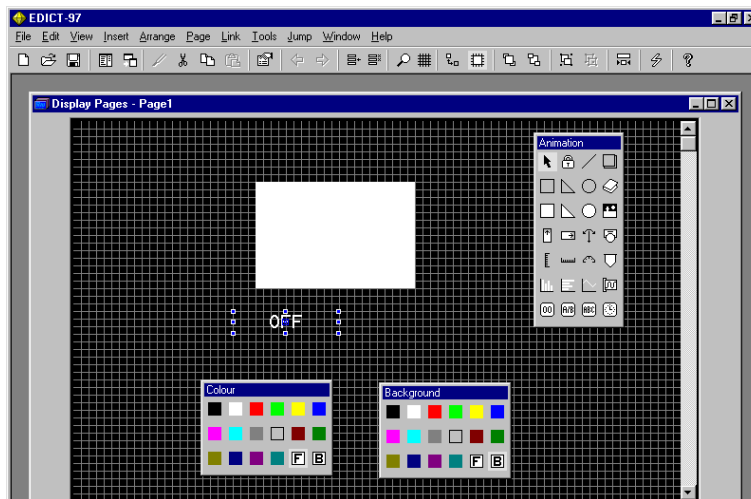
It is also possible to indicate that a color property should assume the value of either the ambient foreground or background color. The default setting for each property is always one of the ambient colors. Setting a property to its default produces no distinct result. Setting a property to the other ambient color is useful, however. One may, for example wish to define a text field whose background is equal to the ambient foreground color and whose foreground color is equal to the ambient background. Such a field will always appear "in reverse"; even if the page's foreground or background color is changed.

### Selecting the color of an animation item from the two color toolboxes



The inserted rectangle has one color property that can be set. Therefore the foreground window "becomes active" and the color of this item can be changed directly by using your mouse to choose a color from this toolbox.

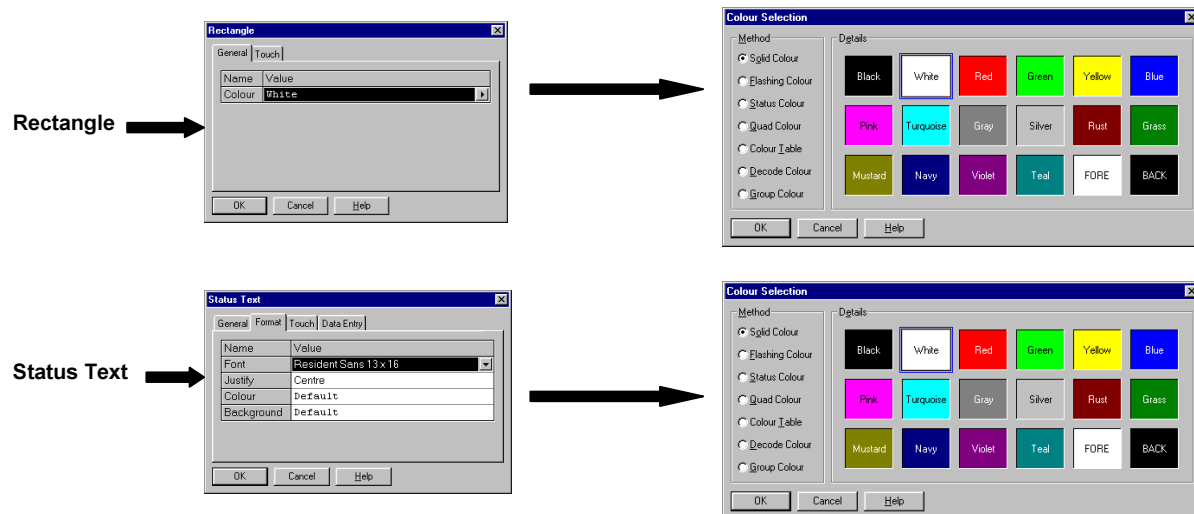
Insert an animation item that has both foreground and background properties



The Status Text item inserted on the display page has both foreground and background colors which can be selected. Notice that the background toolbox now becomes active. Both background and foreground colors can be selected by moving your mouse and clicking the desired color.

Both toolboxes are now active

You can also edit the color of an animation item by double clicking it to edit the item's properties



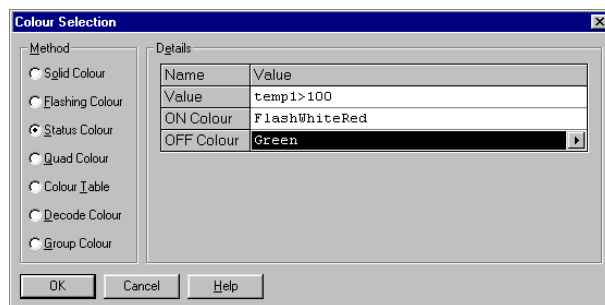
Note: To select the color of the Status Text animation item, the **Format** Tab must be selected.

Another way of defining a color is to use one of the four color animation primitives. These allow a color property to be defined in an analogous way to the various text animation items. For example, the status color primitive allows one of two colors to be chosen based upon a control value, while the quad color primitive extends this to four colors. Further primitives exist to allow a color to be chosen from a list according to a numeric index, or on the basis of a number of conditions. These operate in a manner analogous to the message and text fields, respectively.

### Status Color

Value	An integer expression used to select the color to be displayed. Note that only the truth or otherwise of the expression is considered, with any non-zero value being considered <i>true</i> .
On Color	The color to be displayed when “Value” is <i>true</i> .
Off Color	The color to be displayed when “Value” is <i>false</i> .

### Status Color example

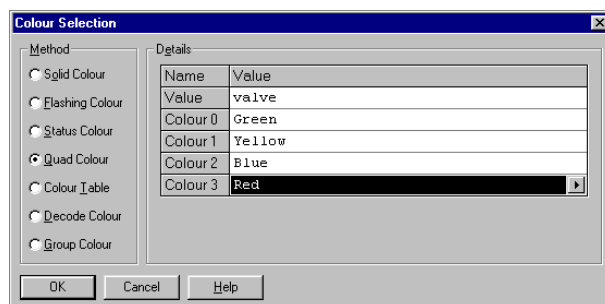


The color of the animation item will be FlashingWhiteRed when the value of temp1 is greater than 100. The color of the animation item will be green when the value of temp1 is less than or equal to 100.

### Quad Color

Value	An integer expression used to select the color to be displayed.
Color 1	The color to display when the bottom bits of “Value” are 00.
Color 2	The color to display when the bottom bits of “Value” are 01.
Color 3	The color to display when the bottom bits of “Value” are 10.
Color 4	The color to display when the bottom bits of “Value” are 11.

### Quad Color example

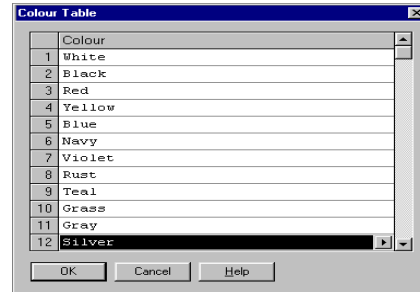
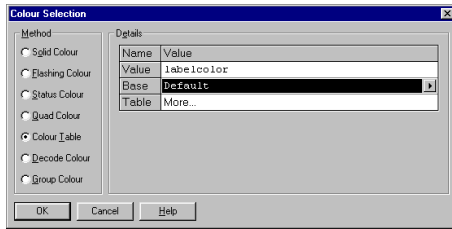


The color of the animation item will be Green when the bottom bits of valve are 00, Yellow when the bottom bits of valve are 01, Blue when the bottom bits of valve are 10 and Red when the bottom bits of valve are 11.

## Color Table

Value	An integer expression defining the color to be displayed.
Base	The color to show when “Value” evaluates to zero. Leaving this property at “Default” will result in an invalid color being selected in such circumstances, with the valid colors starting when “Value” is one. This field is used to introduce an offset into the color selection process, without having to make “Value” into a non-writable value by including the offset within that expression. It is rarely used with a local color table.
Table	Expanding the field allows you to create a color table local to this animation field.

## Color Table example

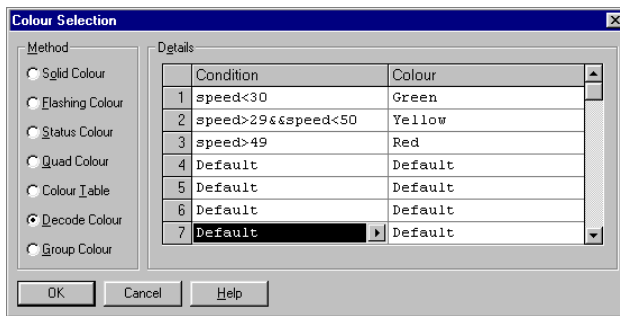


When the value of labelcolor is 1, the color of the animation item is white. When the value of labelcolor is 2, the color of the animation item is black, etc.

## Decode Color

Table	The decode color of this animation item. Expand the field and then enter pairs of controlling expressions and colors. EDICT-97 will display the first color for which the controller expression is <i>true</i> , or left empty. The order of the entries in the table is obviously very important.
-------	--

## Decode Color example



The color of the animation item will be Green when the variable speed is less than 30. The color of the animation item will be Yellow when the value of speed is greater than 29 and less than 50. The color of the animation item will be Red when the value of speed is greater than 49.

## Group Color (See Using Group Properties in Macros Page E14)

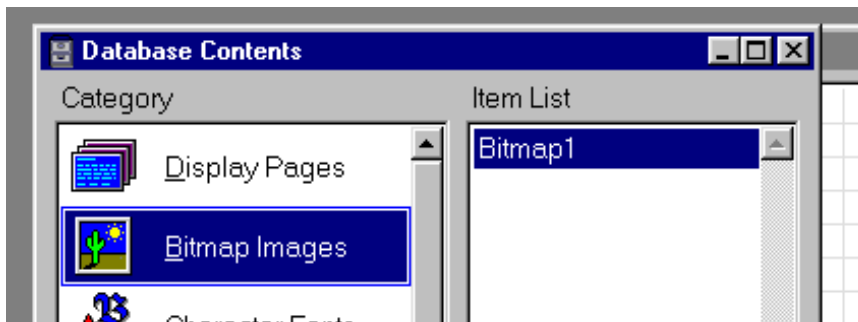
## Bitmap Animation Item

This animation item displays a bitmap from the Bitmap Images list. If the bitmap has multiple sub-images, a value can be used to select from those images. This technique can be used to create animation items, which display one of a number of bitmaps based upon the status of the plant.

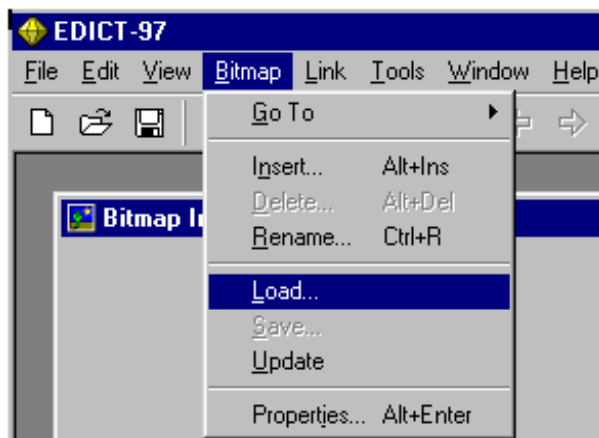
The table below lists the properties of this animation item...

Property (General)	Description
Value	The sub-image to be displayed. A value of zero displays the first sub-image, and a value of one displays the second, and so on. If the bitmap does not have sub-images, this value has no effect on what is displayed. If the value exceeds that permitted by the number of sub-images, it is reduced modulo that number.
Bitmap	The bitmap to be displayed. The name refers to an entry in the Bitmap Images list. The bitmap must be of a type suitable for the currently selected terminal and display format.
Property (Format)	Description
Color	Refers to the foreground color of an imported monochrome bitmap used with color graphical units. See Color Selection Table for options (Page E41)
Background	Refers to the background color of an imported monochrome bitmap used with color graphical units. See Color Selection Table for options (Page E41)
Property(Touch) (Touch Units Only)	Description
Touch Sensitive	Defines whether this bitmap animation item responds to a touch stimulus. Options are Yes or No.
Enable	Can be used when the Touch Sensitive property is Yes. If an expression is entered for this property, it must evaluate to a non-zero value for this Integer to be Touch Sensitive.
Access Level	Used with the System Security feature to control user access. Access level can be set from "Any" up through "Level 9".
On Pressed	The Action that results when a touch occurs on a Touch Sensitive area. Example: motors.0:=!motors.0 (when the bitmap item is touched, the least significant bit of the variable motors is toggled)
On Auto-Repeat	The Action that results when a touch is maintained on a Touch Sensitive area. Example: motors.0:=!motors.0 (when the touch to the bitmap item is maintained, the least significant bit of the variable motors is toggles).
On Released	The Action that results when a touch occurs and is removed on a Touch Sensitive area. Example: motors.0:=!motors.0 (when the touch to the bitmap item is released, the least significant bit of the variable motors is toggled).

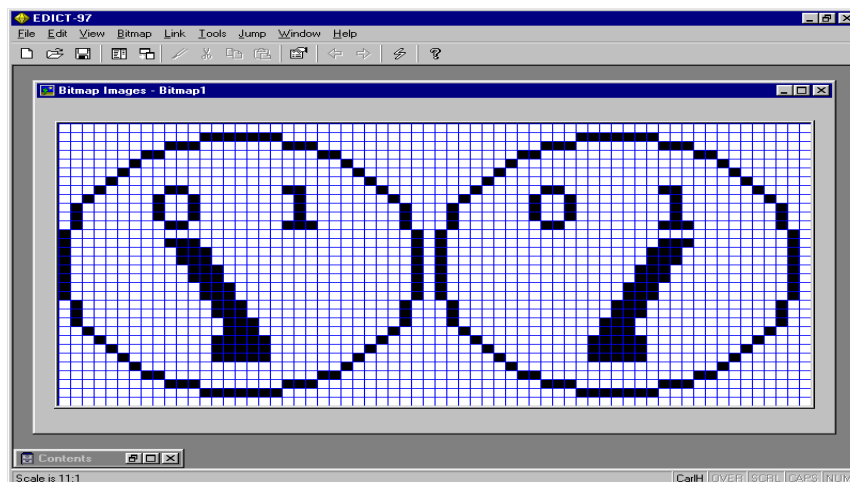
To load custom **BITMAPS** for use in displays, first select "Bitmap Images" from the main screen.



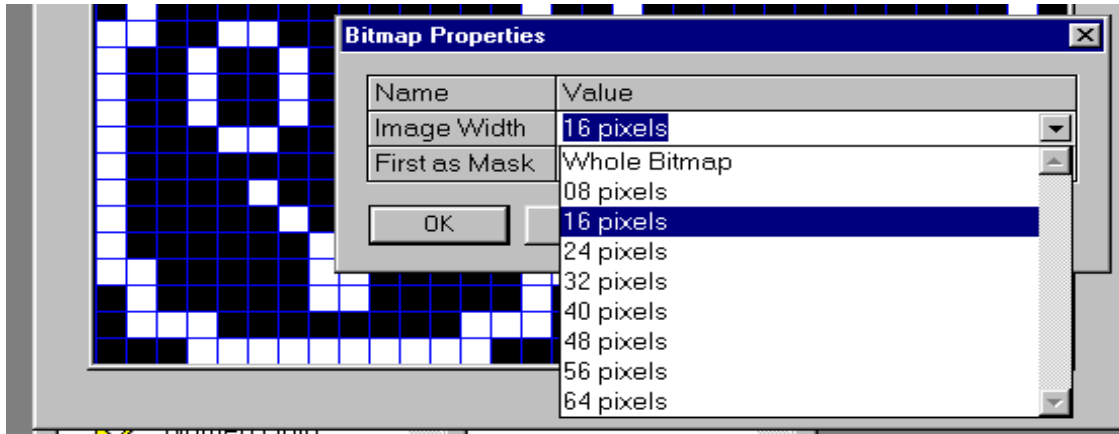
The first bitmap can be loaded by selecting Load. It can be renamed from the default name "Bitmap 1". Subsequent bitmaps can be loaded by first selecting "Insert", then named, and loaded. "GoTo" permits the selection of previously loaded bitmaps; "Update" reloads the bitmap from the disk, making it easier to reload when the bitmap was modified.



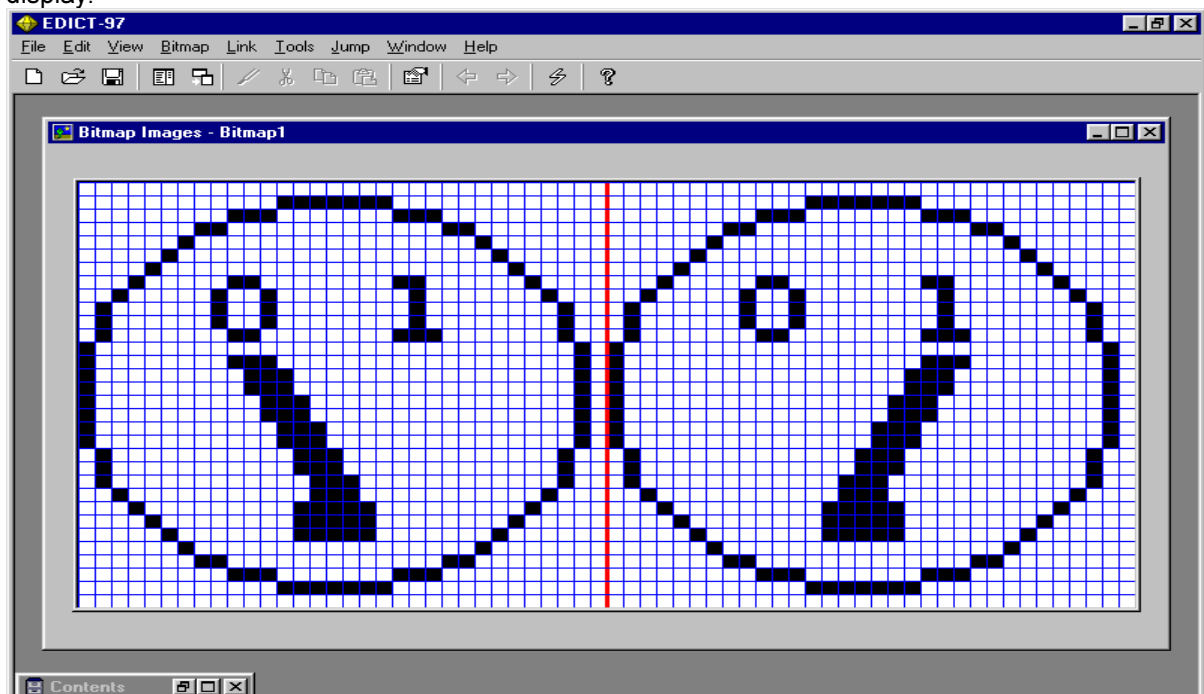
The above is a representation of a rotary switch with 2 states.



By selecting “Bitmap” and “Properties”, the bitmap is divided in half. The terminal’s internal software recognizes the division and will select the left half when the “State” of the animation item is 0, and the right half when the “State” is 1. Since this image is 32 pixels wide, 16 pixels is selected for the display of 2 states. In order to use this property of the software, bitmap sections must be multiples of 8 from 16 through 64, and the entire bitmap must be a multiple of a section.

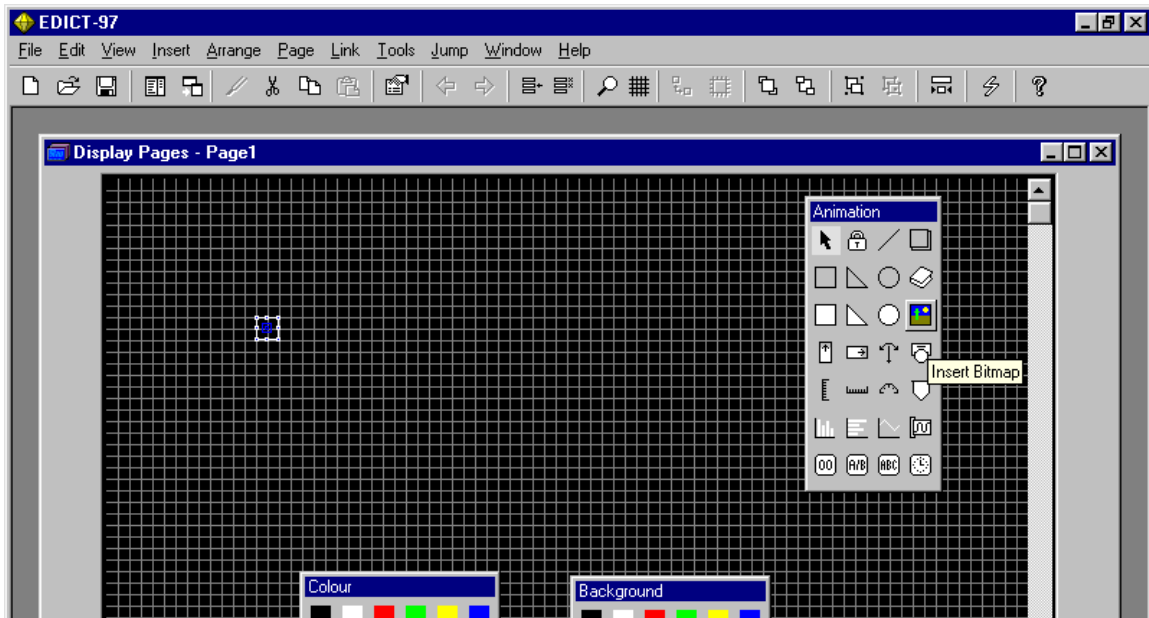


Now the Bitmap is loaded and sectioned. Returning to Display Pages, it can be selected for the display.

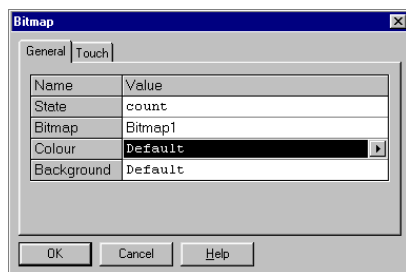


**Note:** Designers can use this bitmap sectioning technique to turn imported 16 color bitmaps “on” and “off”. For example, a 128 x 64 pixel color bitmap can be imported where the object to be displayed is located in the first 64 pixels and next 64 pixels are left blank.

Select **Insert Bitmap** from animation toolbox (this example shows the Graphics layer for a VX color panel) and place on display page

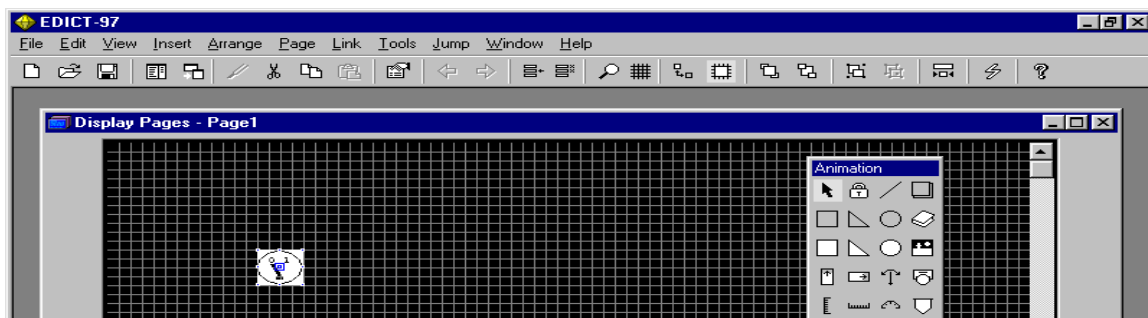


Double click to bitmap animation item to edit properties



The value of the variable count will control the state of the inserted bitmap. The loaded Bitmap is Bitmap 1.

Once OK is selected; the final display is as follows. The size of the item is proportional to the size of the bitmap. To change the size, the original bitmap must be enlarged or reduced.



## Integer Animation Item (Graphics Mode)

This animation item is used to display an integer value on the screen. You may specify how many decimal places are to be shown, and what number base is to be used when formatting the data. The table below lists the properties of this animation item...

Property(General)	Description
Value	An integer expression giving the value to be displayed.
Transform	An optional transform to be performed on the data. Before the data value is displayed, it will be transformed using the information in this property. This can be used, for example, to scale values from PLC to engineering units. Note that the "Value" property is always evaluated as a 32-bit number, and that transforms that manipulate bits will thus behave as if they have been passed a 32-bit argument, even if the underlying data is really a 16-bit value.
Template	A picture of how the number should be formatted. You should use this field to select the number of decimal places required and the number base to be used. You may either select a value from the list, or enter a custom setting of your own. Follow the link below for more details on using Templates.
Mode	Read-only or Data-Entry

Property(Format)	Description
Font	The font to be used to display the text. You can either select one of the six resident fonts present in the terminal's firmware, or select a font defined in the Character Fonts section.
Justify	How the message is to be formatted. This setting controls how EDICT-97 pads-out graphical text, which is shorter than the width of the animation item.
Color	The Color to be used for the text. See Color Selection Table for options (Page E41).
Background	The Color to be used for the background. See Color Selection Table for options (Page E41).

Property(Touch) (Touch Units Only)	Description
Touch Sensitive	Defines whether this Inserted Integer field responds to a touch stimulus. Options are Yes or No.
Enable	Can be used when the Touch Sensitive property is Yes. If an expression is entered for this property, it must evaluate to a non-zero value for this Integer to be Touch Sensitive.
Access Level	Used with the System Security feature to control user access. Access level can be set from "Any" up through "Level 9".
On Pressed	The Action that results when a touch occurs on a Touch Sensitive area. Example: <b>count:=914</b> (When the Integer value field is pressed the value "count" will be loaded with the value 914).

On Auto-Repeat	The Action that results when a touch is maintained on a Touch Sensitive area. Example: count++ (the register “count” will continuously increment by 1 as long as the touch is maintained).
On Released	The Action that results when a touch occurs and is removed on a Touch Sensitive area. Example: <b>count:=914</b> (When the Integer value field is pressed, the value “count” will be loaded with the value 914).

---

**Property(DataEntry)  
(Touch Units Only)**

	<b>Description</b>
Minimum	The minimum value permitted. If an expression is entered, the value entered by the operator must not be less than the value of the expression. If an out of range value is entered, the “Out of Range Value Entered” event will be generated.
Maximum	The maximum value permitted. If an expression is entered, the value entered by the operator must not be more than the value of the expression. If an out of range value is entered, the “Out of Range Value Entered” event will be generated.
Step	The step used for the Raise and Lower keys. This field is used to control how much the value will change by when the Raise and Lower keys are used for data entry. Leaving the field at its default value will produce a step of one, while entering a zero value will effectively disable these keys.
Default	The default value for the field. If an expression is entered, the field will be loaded with the value of that expression before data entry is commenced. If the property is left at its default value, the field will contain whatever was previously held in the expression used to define its “Value” property.
Enable	Whether or not data entry is permitted. If an expression is entered for this property, it must evaluate to a non-zero value if data entry is to be allowed. If the expression is zero, the field will operate as if the “Mode” setting had been set to “read-only”.
Access Level	Used with the System Security feature to control user access.  Access level can be set from “Any” up through “Level 9”.
Touch Entry	Touch units feature a “popup” data-entry window when data entry fields are touched. The four options for these windows are Raise-Lower and Keypad, Raise-Lower Only, Keypad Only and Just Select field.

---

**Real Number Animation Item (Graphics Mode)**

This animation item is used to display a floating point value on the screen or on a printed report. You may specify how many decimal places are to be shown by modifying the template. Note that only fixed-point formatting of floating point values is supported. The table below lists the properties of this animation item...

<b>Property (General)</b>	<b>Description</b>
Value	A floating point expression giving the value to be displayed. If you want to display an integer value, you should consider using the Integer animation item instead, as it is somewhat quicker and supports a greater number of formatting options.
Template	A picture of how the number should be formatted. You should use this field to select the number of decimal places required. You may either select a value from the list, or enter a custom setting of your own. Follow the link below for more details on using Templates.

<b>Property (Format)</b>	<b>Description</b>
Font	The font to be used to display the text. You can either select one of the six resident fonts present in the terminal's firmware, or select a font defined in the Character Fonts section.
Justify	How the message is to be formatted. This setting controls how EDICT-97 pads-out graphical text, which is shorter than the width of the animation item.
Color	The Color to be used for the text. See Color Selection Table for options (Page E41).
Background	The Color to be used for the background. See Color Selection Table for options (Page E41).

<b>Property(Touch) (Touch Units Only)</b>	<b>Description</b>
Touch Sensitive	Defines whether this Inserted Integer field responds to a touch stimulus. Options are Yes or No.
Enable	Can be used when the Touch Sensitive property is Yes. If an expression is entered for this property, it must evaluate to a non-zero value for this Integer to be Touch Sensitive.
Access Level	Used with the System Security feature to control user access. Access level can be set from "Any" up through "Level 9".
On Pressed	The Action that results when a touch occurs on a Touch Sensitive area. Example: <b>count:=2.5</b> (When the Real value field is pressed, the value "count" will be loaded with the value 2.5). Example: count:=2.5 (The register "count" contains a value of 2.5 when the Value is touched).
On Auto-Repeat	The Action that results when a touch is maintained on a Touch Sensitive area. Example: count=count+2.5 (the register "count" will continuously increment by 2.5 as long as the touch is maintained).
On Released	The Action that results when a touch occurs and is removed on a Touch Sensitive area. Example: count:=2.5 (The register "count" contains a value of 2.5 when the Real Value area is touched & released).

### Fixed Text Animation Item (Graphics Mode)

This animation item displays a fixed string.

The table below lists the properties of this animation item...

Property (General)	Description
Text	A string specifying the text to be displayed.
Length	The text string length can be specified. For Example, entering a 5 in this field would limit the text string length to 5 characters. Leaving the setting in this field at "Automatic" allows for string lengths limited only by the display page layout.

Property (Format)	Description
Font	The font to be used to display the text. You can either select one of the six resident fonts present in the terminal's firmware, or select a font defined in the Character Fonts section.
Justify	How the message is to be formatted. This setting controls how EDICT-97 truncates graphical text, which is shorter than the width of the animation item.
Color	The Color to be used for the text. See Color Selection Table for options (Page E41).
Background	The Color to be used for the background. See Color Selection Table for options (Page E41).

Property(Touch) (Touch Units Only)	Description
Touch Sensitive	Defines whether this Fixed field responds to a touch stimulus. Options are Yes or No.
Enable	Can be used when the Touch Sensitive property is Yes. If an expression is entered for this property, it must evaluate to a non-zero value for this Integer to be Touch Sensitive.
Access Level	Used with the System Security feature to control user access. Access level can be set from "Any" up through "Level 9".
On Pressed	The Action that results when a touch occurs on a Touch Sensitive area. Example: <b>count:=914</b> (When the Integer value field is pressed, the value "count" will be loaded with the value 914).
On Auto-Repeat	The Action that results when a touch is maintained on a Touch Sensitive area. Example: <b>count++</b> (the register "count" will continuously increment by 1 as long as the touch is maintained).
On Released	The Action that results when a touch occurs and is removed on a Touch Sensitive area. . Example: <b>count:=914</b> (When the Integer value field is pressed, the value "count" will be loaded with the value 914).

### Status Text Animation Item (Graphics Mode)

The animation item is used to display one of a pair of text strings, based upon the logical value of an expression. For example, it can be used to display the status of a single bit, or to display a string based upon whether a numeric value is over a certain limit. The table below lists the properties of this animation item...

Property (General)	Description
Value	An integer expression used to select the text to be displayed. Note that only the truth or otherwise of the expression is considered, with any non-zero value being considered <i>true</i> .
ON Text	The text to display when "Value" is <i>true</i> .
OFF Text	The text to display when "Value" is <i>false</i> .
Length	The text string length can be specified. For Example, entering a 5 in this field would limit the text string length to 5 characters. Leaving the setting in this field at "Automatic" allows for string lengths limited only by the display page layout.
Mode	Data-Entry or Read-Only. On Touch units the Raise/Lower data entry window will appear when the Status Text area is touched. On non-touch Graphic units, the Raise/Lower keys will toggle the Status Text. The Integer "Value" will also change when the Status Text changes.

Property (Format)	Description
Font	The font to be used to display the text. You can either select one of the six resident fonts present in the terminal's firmware, or select a font defined in the Character Fonts section.
Justify	How the message is to be formatted. This setting controls how EDICT-97 pads-out graphical text, which is shorter than the width of the animation item.
Color	The Color to be used for the text. See Color Selection Table for options (Page E41).
Background	The Color to be used for the background. See Color Selection Table for options (Page E41).

Property(Touch) (Touch Units Only)	Description
Touch Sensitive	Defines whether this Status Text field responds to a touch stimulus. Options are Yes or No.
Enable	Can be used when the Touch Sensitive property is Yes. If an expression is entered for this property, it must evaluate to a non-zero value for this Integer to be Touch Sensitive.
Access Level	Used with the System Security feature to control user access. Access level can be set from "Any" up through "Level 9".
On Pressed	The Action that results when a touch occurs on a Touch Sensitive area. Example: lang:=!lang,SetLanguage(lang) (the language that the database is displayed in will toggle each time that the Status Text field is touched.)

On Auto-Repeat	The Action that results when a touch is maintained on a Touch Sensitive area.
On Released	The Action that results when a touch occurs and is removed on a Touch Sensitive area. Example: count:=2.5 (The register "count" contains a value of 2.5 when the Status Text area is touched & released).

---

**Property(DataEntry)  
(Touch Units Only)**

**Description**

Default	The default value for the field. If an expression is entered, the field will be loaded with the value of that expression before data entry is commenced. If the property is left at its default value, the field will contain whatever was previously held in the expression used to define its "Value" property.
Enable	Whether or not data entry is permitted. If an expression is entered for this property, it must evaluate to a non-zero value if data entry is to be allowed. If the expression is zero, the field will operate as if the "Mode" setting had been set to "read-only".
Access Level	Used with the System Security feature to control user access. Access level can be set from "Any" up through "Level 9".

---

**Quad Text Animation Item (Graphics Mode)**

The animation item is used to display one of four text strings, based upon the value of the bottom two bits of an expression. It is often used to display the status of a three way valve, based upon the limit switches at either end of its travel. The table below lists the properties of this animation item...

<b>Property (General)</b>	<b>Description</b>
Value	An integer expression used to select the text to be displayed.
Text 0	The text to display when the bottom bits of "Value" are 00.
Text 1	The text to display when the bottom bits of "Value" are 01.
Text 2	The text to display when the bottom bits of "Value" are 10.
Text 3	The text to display when the bottom bits of "Value" are 11.
Length	The text string length can be specified. For Example, entering a 5 in this field would limit the text string length to 5 characters. Leaving the setting in this field at "Automatic" allows for string lengths limited only by the display page layout.
Mode	Data-Entry or Read-Only. On Touch units the Raise/Lower data entry window will appear when the Quad Text area is touched. On non-touch Graphic units, the Raise/Lower keys will alter the Quad Text. The Integer "Value" will also change when the Quad Text changes.

---

<b>Property (Format)</b>	<b>Description</b>
Font	The font to be used to display the text. You can either select one of the six resident fonts present in the terminal's firmware, or select a font defined in the Character Fonts section.
Justify	How the message is to be formatted. This setting controls how EDICT-97 pads-out graphical text, which is shorter than the width of the animation item.
Color	The Color to be used for the text. See Color Selection Table for options (Page E41).
Background	The Color to be used for the background. See Color Selection Table for options (Page E41).

<b>Property(Touch) (Touch Units Only)</b>	<b>Description</b>
Touch Sensitive	Defines whether this Quad Text field responds to a touch stimulus. Options are Yes or No.
Enable	Can be used when the Touch Sensitive property is Yes. If an expression is entered for this property, it must evaluate to a non-zero value for this Integer to be Touch Sensitive.
Access Level	Used with the System Security feature to control user access. Access level can be set from "Any" up through "Level 9".
On Pressed	The Action that results when a touch occurs on a Touch Sensitive area.
On Auto-Repeat	The Action that results when a touch is maintained on a Touch Sensitive area.
On Released	The Action that results when a touch occurs and is removed on a Touch Sensitive area

<b>Property(DataEntry) (Touch Units Only)</b>	<b>Description</b>
Minimum	The minimum value permitted. If an expression is entered, the value entered by the operator must not be less than the value of the expression. If an out of range value is entered, the "Out of Range Value Entered" event will be generated.
Maximum	The maximum value permitted. If an expression is entered, the value entered by the operator must not be more than the value of the expression. If an out of range value is entered, the "Out of Range Value Entered" event will be generated.
Step	The step used for the Raise and Lower keys. This field is used to control how much the value will change by when the Raise and Lower keys are used for data entry. Leaving the field at its default value will produce a step of one, while entering a zero value will effectively disable these keys.

Default	The default value for the field. If an expression is entered, the field will be loaded with the value of that expression before data entry is commenced. If the property is left at its default value, the field will contain whatever was previously held in the expression used to define its "Value" property.
Access Level	Used with the System Security feature to control user access. Access level can be set from "Any" up through "Level 9".

---

### Message Text Animation Item (Graphics Mode)

This animation item is used to display a text string chosen from the message table, with the selection being based upon the value of an expression. You can select from either the global message table, or a table local to a given animation item. The table below lists the properties of this animation item...

Property (General)	Description
Value	An integer expression defining the message to be displayed.
Base	The message to show when "Value" evaluates to zero. Leaving this property at "Default" will result in an invalid message being selected in such circumstances, with the valid messages starting when "Value" is one. This field is used to introduce an offset into the message selection process, without having to make "Value" into a non-writable value by including the offset within that expression. It is rarely used with a local message table.
Table	The message table to use. If you leave this property set to "Global", the message will be selected from the global message table. Expanding the field allows you to create a message table local to this animation field. This can make life easier sometimes, but can be wasteful of memory and harder to maintain if the same table is to be repeated many times.
Length	The length of the field on the display page. You should enter a value from 1 to the maximum number of characters permitted by the display page format and the cursor position. Messages longer than this value will be clipped according to the "Justify" setting.
Mode	Data-Entry or Read-Only. On Touch units, the Raise/Lower data entry window will appear when the Message Text area is touched. On non-touch Graphic units, the Raise/Lower keys will alter the Quad Text. The Integer "Value" will also change when the Message Text changes.

---

Property (Format)	Description
Font	The font to be used to display the text. You can either select one of the six resident fonts present in the terminal's firmware, or select a font defined in the Character Fonts section.
Justify	How the message is to be formatted. This setting controls how EDICT-97 pads-out graphical text, which is shorter than the width of the animation item.

Color	The Color to be used for the text. See Color Selection Table for options (Page E41).
Background	The Color to be used for the background See Color Selection Table for options (Page E41).

<b>Property(Touch) (Touch Units Only)</b>	<b>Description</b>
Touch Sensitive	Defines whether this Message Text field responds to a touch stimulus. Options are Yes or No.
Enable	Can be used when the Touch Sensitive property is Yes. If an expression is entered for this property, it must evaluate to a non-zero value for this Integer to be Touch Sensitive.
Access Level	Used with the System Security feature to control user access. Access level can be set from "Any" up through "Level 9".
On Pressed	The Action that results when a touch occurs on a Touch Sensitive area. Example: count:=2.5 (The register "count" contains a value of 2.5 when the Message Text area is touched).
On Auto-Repeat	The Action that results when a touch is maintained on a Touch Sensitive area. Example: count:=count +2.5 (the register "count" will continuously increment by 2.5 as long as the touch is maintained).
On Released	The Action that results when a touch occurs and is removed on a Touch Sensitive area. Example: count:=2.5 (The register "count" contains a value of 2.5 when the Message Text area is touched & released).

<b>Property(DataEntry) (Touch Units Only)</b>	<b>Description</b>
Minimum	The minimum value permitted. If an expression is entered, the value entered by the operator must not be less than the value of the expression. If an out of range value is entered, the "Out of Range Value Entered" event will be generated.
Maximum	The maximum value permitted. If an expression is entered, the value entered by the operator must not be more than the value of the expression. If an out of range value is entered, the "Out of Range Value Entered" event will be generated.
Step	The step used for the Raise and Lower keys. This field is used to control how much the value will change by when the Raise and Lower keys are used for data entry. Leaving the field at its default value will produce a step of one, while entering a zero value will effectively disable these keys.
Default	The default value for the field. If an expression is entered, the field will be loaded with the value of that expression before data entry is commenced. If the property is left at its default value, the field will contain whatever was previously held in the expression used to define its "Value" property.
Access Level	Used with the System Security feature to control user access. Access level can be set from "Any" up through "Level 9".

## Decode Text Animation Item (Graphics Mode)

This animation item is used to select a string from a table, based upon the *true* or *false* value of an expression associated with each string. EDICT-97 scans the table, and selects the first string for which the controlling expression is *true*. This allows complex decoding functions to be performed, including such things as bit-level prioritisation of messages, or multiple decodes of numeric values. The table below lists the properties of this animation item...

Property (General)	Description
Table	The decode table of this field. You should expand the field by pressing the F2 key, and then enter pairs of controlling expressions and strings. EDICT-97 will display the first string for which the controller expression is <i>true</i> , or left empty. The order of the entries in the table is obviously very important.

Property (Format)	Description
Length	The length of the field on the display page. You should enter a value from 1 to the maximum number of characters permitted by the display page format and the cursor position. Strings longer than this value will be clipped according to the "Justify" setting.
Font	The font to be used to display the text. You can either select one of the six resident fonts present in the terminal's firmware, or select a font defined in the Character Fonts section.
Justify	How the message is to be formatted. This setting controls how EDICT-97 pads-out graphical text, which is shorter than the width of the animation item.
Color	The Color to be used for the text. See Color Selection Table for options (Page E41).
Background	The Color to be used for the background. See Color Selection Table for options (Page E41).

Property(Touch) (Touch Units Only)	Description
Touch Sensitive	Defines whether this Decode Text field responds to a touch stimulus. Options are Yes or No.
Enable	Can be used when the Touch Sensitive property is Yes. If an expression is entered for this property, it must evaluate to a non-zero value for this Integer to be Touch Sensitive.
Access Level	Used with the System Security feature to control user access. Access level can be set from "Any" up through "Level 9".
On Pressed	The Action that results when a touch occurs on a Touch Sensitive area. Example: count:=2.5 (The register "count" contains a value of 2.5 when the Decode Text area is touched).

On Auto-Repeat	The Action that results when a touch is maintained on a Touch Sensitive area. Example: count:=count +2.5 (the register "count" will continuously increment by 2.5 as long as the touch is maintained).
On Released	The Action that results when a touch occurs and is removed on a Touch Sensitive area. Example: count:=2.5 (The register "count" contains a value of 2.5 when the Decode Text area is touched & released).

### General Text Animation Item (Graphics Mode)

This animation item displays the value of a string expression. As well as displaying string data from the PLC or other comms devices, it can be used together with the "CallString" function to implement custom animation types. For example, you might run a program which examines level data, and then either returns the value formatted using the "Format" function, or an indication that the level is too high or low. The table below lists the properties of this animation item...

Property (General)	Description
Value	A string expression giving the text to be displayed.
Length	The length of the field on the display page. You should enter a value from 1 to the maximum number of characters permitted by the display page format and the cursor position. Strings longer than this value will be clipped according to the "Justify" setting.
Mode	Data-Entry or Read-Only. On Touch units, the Raise/Lower data entry window will appear when the General Text area is touched. On non-touch Graphic units, the Raise/Lower keys will alter the General Text.

Property (Format)	Description
Font	The font to be used to display the text. You can either select one of the six resident fonts present in the terminal's firmware, or select a font defined in the Character Fonts section.
Justify	How the message is to be formatted. This setting controls how EDICT-97 pads-out graphical text, which is shorter than the width of the animation item.
Color	The Color to be used for the text. See Color Selection Table for options (Page E41).
Background	The Color to be used for the background. See Color Selection Table for options (Page E41).

Property(Touch) (Touch Units Only)	Description
Touch Sensitive	Defines whether this General Text field responds to a touch stimulus. Options are Yes or No.
Enable	Can be used when the Touch Sensitive property is Yes. If an expression is entered for this property, it must evaluate to a non-zero value for this Integer to be Touch Sensitive.

Access Level	Used with the System Security feature to control user access. Access level can be set from “Any” up through “Level 9”.
On Pressed	The Action that results when a touch occurs on a Touch Sensitive area. Example: count:=2.5 (The register “count” contains a value of 2.5 when the General Text area is touched).
On Auto-Repeat	The Action that results when a touch is maintained on a Touch Sensitive area. Example: count:=count +2.5 (the register “count” will continuously increment by 2.5 as long as the touch is maintained).
On Released	The Action that results when a touch occurs and is removed on a Touch Sensitive area. Example: count:=2.5 (The register “count” contains a value of 2.5 when the General Text area is touched & released).

### Time & Date Animation Item (Graphics Mode)

This animation item is used to display a time or date.

Note that the time is always shown in 24-hour or military format.

The table below lists the general properties of this animation item...

Property (General)	Description
Value	The time and date to be displayed. If left as “Default”, the current time and date will be shown. Otherwise, the value will be taken as a 32-bit unsigned quantity representing the number of seconds elapsed since midnight on the 1 <sup>st</sup> January 4197. This is the same encoding used by all of EDICT-97’s internal time and date functions.
Template	A string showing how the time or date should be formatted. Each character in the template either represents a character to be literally copied to the output stream, or a placeholder for time or date information. See the table below for details of each placeholder, and what it represents.

### Template Placeholders

The table below lists the possible placeholder characters, and explains their effects...

Character	Description
‘h’	Will be replaced with the next available digit of the current hour.
‘m’	Will be replaced with the next available digit of the current minute.
‘s’	Will be replaced with the next available digit of the current second.
‘Y’	Will be replaced with the next available digit of the current year.
‘M’	Will be replaced with the next available digit of the current month.
‘D’	Will be replaced with the next available digit of the current date.

<b>Property (Format)</b>	<b>Description</b>
Font	The font to be used to display the text. You can either select one of the six resident fonts present in the terminal's firmware, or select a font defined in the Character Fonts section.
Justify	How the message is to be formatted. This setting controls how EDICT-97 pads-out graphical text, which is shorter than the width of the animation item.
Color	The Color to be used for the text. See Color Selection Table for options (Page E41).
Background	The Color to be used for the background. See Color Selection Table for options (Page E41).

<b>Property(Touch) (Touch Units Only)</b>	<b>Description</b>
Touch Sensitive	Defines whether this Time and Date field responds to a touch stimulus. Options are Yes or No.
Enable	Can be used when the Touch Sensitive property is Yes. If an expression is entered for this property, it must evaluate to a non-zero value for this Integer to be Touch Sensitive.
Access Level	Used with the System Security feature to control user access. Access level can be set from "Any" up through "Level 9".
On Pressed	The Action that results when a touch occurs on a Touch Sensitive area. Example: count:=2.5 (The register "count" contains a value of 2.5 when the Time and Date area is touched).
On Auto-Repeat	The Action that results when a touch is maintained on a Touch Sensitive area. Example: count:=count +2.5 (the register "count" will continuously increment by 2.5 as long as the touch is maintained).
On Released	The Action that results when a touch occurs and is removed on a Touch Sensitive area. Example: count:=2.5 (The register "count" contains a value of 2.5 when the Time and Date area is touched & released).

### **The Line Animation Item**

This animation item draws a straight line between two points.  
The table below lists the properties of this animation item...

<b>Property</b>	<b>Description</b>
Color	The Color to be used for the line. Select a Color from the list displayed by the Alt+Down key combination. See Color Selection Table for options (Page E41).

### The Frame Animation Item

This animation item draws a rectangular frame.

The table below lists the properties of this animation item...

Property (General)	Description
Color	The Color to be used for the frame. Select a Color from the list displayed by the Alt+Down key combination. See Color Selection Table for options (Page E41).
Hide Edges	Left, Right, Top or Bottom

### The Rectangle Animation Item

This animation item draws a solid rectangle.

The table below lists the properties of this animation item...

Property	Description
Color	The Color to be used for the rectangle. Select a Color from the list displayed by the Alt+Down key combination. See Color Selection Table for options (Page E41).

Property(Touch) (Touch Units Only)	Description
Touch Sensitive	Defines whether this Rectangle field responds to a touch stimulus. Options are Yes or No.
Enable	Can be used when the Touch Sensitive property is Yes. If an expression is entered for this property, it must evaluate to a non-zero value for this Integer to be Touch Sensitive.
Access Level	Used with the System Security feature to control user access. Access level can be set from "Any" up through "Level 9".
On Pressed	The Action that results when a touch occurs on a Touch Sensitive area. Example: count:=2.5 (The register "count" contains a value of 2.5 when the Rectangle area is touched).
On Auto-Repeat	The Action that results when a touch is maintained on a Touch Sensitive area. Example: count:=count +2.5 (the register "count" will continuously increment by 2.5 as long as the touch is maintained).
On Released	The Action that results when a touch occurs and is removed on a Touch Sensitive area. Example: count:=2.5 (The register "count" contains a value of 2.5 when the Rectangle area is touched & released).

### The Shadow Animation Item

This animation item draws a rectangular frame with a drop shadow.  
The table below lists the properties of this animation item...

Property (General)	Description
Color	The Color to be used for the figure. Select a Color from the list displayed by the Alt+Down key combination. See Color Selection Table for options (Page E41).
Style	Drop Shadow, Raised Border or Sunken Border.

### The Wedge Animation Item

This animation item draws a solid triangle within a specified rectangle.  
The table below lists the properties of this animation item...

Property (General)	Description
Color	The Color to be used for the figure. Select a Color from the list displayed by the Alt+Down key combination. See Color Selection Table for options (Page E41).
Orientation	The position of the wedge within the rectangle. Select the required orientation from the drop-down list by pressing the Alt+Down key combination.

### The Circle Animation Item

This animation item draws an outline circle.  
The table below lists the properties of this animation item...

Property	Description
Color	The Color to be used for the circle. Select a Color from the list displayed by the Alt+Down key combination. See Color Selection Table for options (Page E41).

### Property(Touch) (Touch Units Only)

(Touch Units Only)	Description
Touch Sensitive	Defines whether this Circle field responds to a touch stimulus. Options are Yes or No.
Enable	Can be used when the Touch Sensitive property is Yes. If an expression is entered for this property, it must evaluate to a non-zero value for this Integer to be Touch Sensitive.
Access Level	Used with the System Security feature to control user access. Access level can be set from "Any" up through "Level 9".

On Pressed	The Action that results when a touch occurs on a Touch Sensitive area. Example: count:=2.5 (The register "count" contains a value of 2.5 when the Circle area is touched).
On Auto-Repeat	The Action that results when a touch is maintained on a Touch Sensitive area. Example: count:=count +2.5 (the register "count" will continuously increment by 2.5 as long as the touch is maintained).
On Released	The Action that results when a touch occurs and is removed on a Touch Sensitive area. Example: count:=2.5 (The register "count" contains a value of 2.5 when the Circle area is touched & released).

### The Disk Animation Item

This animation item draws a solid circle.

The table below lists the properties of this animation item...

Property (General)	Description
Color	The Color to be used for the Disk. Select a Color from the list displayed by the Alt+Down key combination. See Color Selection Table for options (Page E41).

Property(Touch) (Touch Units Only)	Description
Touch Sensitive	Defines whether this Disk field responds to a touch stimulus. Options are Yes or No.
Enable	Can be used when the Touch Sensitive property is Yes. If an expression is entered for this property, it must evaluate to a non-zero value for this Integer to be Touch Sensitive.
Access Level	Used with the System Security feature to control user access. Access level can be set from "Any" up through "Level 9".
On Pressed	The Action that results when a touch occurs on a Touch Sensitive area. Example: count:=2.5 (The register "count" contains a value of 2.5 when the Disk area is touched).
On Auto-Repeat	The Action that results when a touch is maintained on a Touch Sensitive area. Example: count:=count +2.5 (the register "count" will continuously increment by 2.5 as long as the touch is maintained).
On Released	The Action that results when a touch occurs and is removed on a Touch Sensitive area. Example: count:=2.5 (The register "count" contains a value of 2.5 when the Disk area is touched & released).

---

# Allen Bradley SLC500

## Application Note

---

This document describes how to configure a Paradigm operator interface terminal to allow communications with an Allen Bradley SLC500. The communications protocol supports access to numeric registers, flags, and control actions. Please read this document carefully before attempting to configure communications with these devices.

**Using DF1 with cable P895005Z**

## Channel 0 Configuration

Current Communication Mode	: SYSTEM
System Mode Driver	:DF1 Full-Duplex
User Mode Driver	:SHUTDOWN
Write Protect	:Disabled
Mode Changes	:Disabled
Mode Attention Character	:\1b
System Mode Character	:S
User Mode Character	:U
Edit Resource/File Owner Timeout	:60 (seconds)
Passthru Link ID	:1

## Channel 0 System Mode Configuration

Communication Driver	:DF1 Full-Duplex
Diagnostic File	:Reserved

Baud Rate	: 9600	Parity	:EVEN
Duplicate Detect	:ENABLED	Error Detect	:BCC
ACK Timeout [x20ms]	:50	NAK Retries	:3
		ENQ Retries	:3
Source ID	:9	Embedded Responses	:AUTO-DETECT

## Paradigm Settings:

## Comms Ports :

2 RS-232 Comms Port	Allen Bradley SLC via DF1	Direct Connection	8E1 9600
---------------------	---------------------------	-------------------	----------

**Using DH-485 with cable P895013Z**

Channel 1 Configuration	
System Mode Driver	:DH-485 Master
Write Protect	:Disabled
Edit Resource/File Owner Timeout	:60 (seconds)
Passthru Link ID	:2
Channel 1 System Mode Configuration	
Communication Driver	:DH-485
Diagnostic File	:Reserved
Baud Rate	:19200 (Note 1)
Node Address	:1
Max Node Address	:31
Token Hold Factor	:1

Note 1: The baud rate must be 19200 for reliable communication.

Paradigm Setting:

Comms Ports :

3 RS-485 Comms Port Allen Bradley SLC via DH-485 Direct Connection 8E1 19200

**Attaching to Micrologix (Must be series C or higher)**

Select Micro FULL-DUPLEX in programming Software.

Using bridge cable to 1761-CBL-PM02:

Paradigm Setting:

Comms Ports:

2 RS-232 Comms Port Allen Bradley SLC via DH-485 Direct Connection 8N1 19200

Using cable P895013Z through the AIC+ module:

Use Port 3 settings on previous page.

Proper communications in a system of a Micrologix, SLC, via AIC modules to Paradigm operator interfaces can be obtained by making the Micrologix node address the highest of the three. If more than one Micrologix is present, see <http://www.ab.com/support> choose Micrologix and look at technical documents 9601 and 10519, to assist in configuring the PLC's

This page intentionally left blank

# Control Technology

## Application Note

---

This document describes how to configure a Paradigm operator interface terminal to allow communications with a Control Technology Automation Controller. The communications protocol supports access to numeric registers, flags, and control actions. Please read this document carefully before attempting to configure communications with these devices.

## Introduction

The EDICT-97 configuration software has been designed to allow the user to enter a Parameter mnemonic and number in a manner that should be familiar to a user of a Control Technology Automation Controller. The driver allows the exchange of data with the Controller.

## Accessing Data

The Control Technology Automation Controller communications protocol allows access to a number of parameters over a serial communications link. The driver described here supports a subset of these parameters and these are given in the table below.

Parameter	Mnemonic	Range	Data Type	Access
Numeric Register	REG	00001..65535	32-bit Signed	Read/Write
Flag	FLAG	01..32	Individual Byte	Read/Write
Start Controller	RUN	--	--	Write
Stop Controller	STOP	--	--	Write
Reset Controller	RST	--	--	Write

Writing 0 to a Flag will clear the flag, while writing 255 (0xFF) will set the flag. Writing any other number to a flag will provide indeterminate results.

## Knowledge of Unit Operation Is Assumed

In all cases, the simple principle of 'pass-through' is maintained: there is no attempt to validate a value in terms of the end use of the unit: both familiarity with the control functions and knowledge of system operation are assumed.

## Communications

Communications with the Control Technology Automation Controller is via an RS-232, point to point link, with default serial communications format of baud rate 9600, 8 data bits, No parity, and 1 stop bit.

The connections details are described in the table below.

Paradigm unit (RS 232 port)	Control Technology Automation Controller (RS232 Modular Jack)
Pin 1 (Tx)	Pin 5 (RxD)
Pin 2 (Rx)	Pin 2 (TxD)
Pin 3 (RTS)	
Pin 4 (CTS)	
Pin 5 (0v)	Pin 3 (GND)

In addition a link must be fitted between Pin 3 (RTS) and Pin 4 (CTS) on the Paradigm unit.

# KEB Frequency Inverter

## Application Note

---

This document describes how to configure a Paradigm operator interface terminal so as to allow communications with a KEB Frequency Inverter. The communications protocol supports access to all parameters and parameter sets. Please read this document carefully before attempting to configure communications with these devices.

## Introduction

The EDICT-97 configuration software has been designed to allow the user to enter a Parameter Set and a Communications Parameter Address in a manner that should be familiar to a user of a KEB Frequency Inverter.

## Accessing Data

Each Parameter in the KEB Frequency Inverter has a communications address. In addition the KEB Frequency Inverter has 8 Parameters Set (0..7). Certain Parameters are programmable, meaning that they exist 8 times in the inverter and can be assigned with different values independent of each other. In order that the parameter in the required set is accessed there is a Bus Parameter Set containing the Parameter Set currently accessible via serial link.

When the user is configuring a Communications Block in EDICT-97, the required parameter is referenced in one of 2 forms:

*s.iii*  
*iii*

where *s* is the required parameter set and *iii* is the Parameter communications address, entered as a Hexadecimal value in the range 0..7FFF.

The driver will change the Bus Parameter Set prior accessing the parameter. It makes sense therefore to group Parameters from the same Parameter Set in the same Communication Block.

Some parameters are read-only and others may only be written to when the drive is disabled. Attempts to write to these, or to access non-existent parameters will be gracefully ignored, communications will not be failed.

## Knowledge Of Unit Operation Is Assumed

In all cases, the simple principle of 'pass-through' is maintained: there is no attempt to validate a value in terms of the end use of the unit: both familiarity with the parameters and knowledge of system operation are assumed.

## Communications

EDICT supports RS232 and RS485 connections with a KEB Frequency Inverter, the connection details being described below:

KEB (9 pole D-Sub male)	Paradigm RS232 port	
3	1	Tx
2	2	Rx
-	3	CTS
-	4	RTS
7	5	0v

In addition a link must be made between connectors 3 (CTS) and 4 (RTS) on the Paradigm Operator Interface.

KEB (9 pole D-Sub male)	Paradigm RS485 port	
5	6	TxA
6	7	TxB
8	8	RxA
9	9	RxB
-	10	0v

In addition a 1k8 resistor must be placed between connectors 9 (RxB) and 10 (0v) on the Paradigm Operator Interface.

The default serial communication parameters are as follows, baud rate of 9600, 7 data bits, even parity, and 1 stop bit.

# Panasonic (Minas Series) AC Servo Drive

## Application Note

---

This document describes how to configure a Paradigm operator interface terminal to allow communications with a Panasonic Minas Series AC Servo Drive. It describes the parameters that can be accessed and the controls actions that can be issued to the Drive. Please read this document carefully before attempting to configure communications with these devices.

## Introduction

The EDICT-97 configuration software has been designed to allow the user to enter a Parameter mnemonic and Number if applicable in a manner that should be familiar to a user of a Panasonic Minas Series AC Servo Drive. The driver allows the exchange of data with the Drive, and control actions to be issued to the Drive.

## Accessing Data

The Minas Series communications protocol allows access to a number of Parameters over a serial communications link. The driver described here supports a subset of these parameters and these are given in the table below.

Parameter	Range	Description
User	00..3E	Servo Parameter
System	00..08	Servo Parameter
Step	01..28	NC Parameters – Step Data
Velocity	00..15	NC Parameters – Velocity Data
Data	00..12	NC Parameters – NC Data
Offset	n/a	NC Parameters –Offset Data
Input	n/a	NC Parameters – Input Port
Output	n/a	NC Parameters – Output Port
Position	n/a	Current Position
Speed	n/a	Current Speed
Torque	n/a	Current Torque
Error	n/a	Current Position Error
Alarm	0..6	Alarm Contents
Hist	1..12	Alarm History
CmdError	n/a	Last <i>DeviceCommand</i> Error

In addition, the Step Data is composed of three sub elements, Position Data, Speed Select No, and Positioning Mode Select. These are accessible individually.

In addition, the Offset Data is composed of four sub elements, Origin Offset, Positive Software Limit, Negative Software Limit, and Auxiliary Information 1 and 2. These are accessible individually.

In addition CmdError gives the most recent error reported by the Minas Series Drive in response to a *DeviceCommand* request. The value gives the Error in the low byte and the Command in the high byte, of the low word.

## Control Actions

Often control actions are required to be executed in a particular order in the Drive. EDICT 97 provides the function *DeviceCommand()* which when called will guarantee the command is issued to the Drive in the order that they occur in the terminal. It has the following syntax :

*DeviceCommand(device, "command", parameter)*

where :

*device* is the device number from the Device Table

*command* is the is a string interpreted in the following way

*parameter* is the value associated with the Command.

There are three distinct command string formats, described below. In all cases the C is the command, and M is the Mode, in Hexadecimal format.

*DeviceCommand( D, "CMn", P)*

When the third character is lower-case *n*, the parameter number *P* is included in the command issued to the Drive. This is most applicable for a Step Command where the parameter is the Step Number.

For example, the following command *DeviceCommand( 1, "60n", 3)*, would result in the Stepping command with step No 3 being sent to the Minas Drive on device 1.

*DeviceCommand( D, "CM123", P)*

When the command and more characters are followed by any number of numerical characters, these will appear in the command exactly as entered. This is most applicable for the Jog command.

For example, the following command *DeviceCommand( 1, "5001", 0)*, would result in the Jog command with Jogging Low Speed in the (+) direction being sent to the Minas Drive on device 2. The parameter number *P* is ignored.

*DeviceCommand( D, "CM", P)*

When only the Command and Mode characters appear in the string, the simplest command is sent to the drive. This is most applicable for commands such as the Stop or Org command.

For example, the following command *DeviceCommand( 5, "30", 0)*, would result in the Stop command being sent to the Minas Drive on device 2. The parameter number *P* is ignored.

## Knowledge of Unit Operation Is Assumed

In all cases, the simple principle of 'pass-through' is maintained: there is no attempt to validate a value in terms of the end use of the unit: both familiarity with the control functions and knowledge of system operation are assumed.

## Communications

Communications with the Minas Series AC Drive is via an RS-232, point to point link, with default serial communications format of baud rate of 9600, 8 data bits, No parity, and 1 stop bit.

The connections details are described in the table below.

Paradigm unit (RS 232 port)	Minas Series Drive (MNI DIN8)
Pin 1 (Tx)	Pin 5 (RXD)
Pin 2 (Rx)	Pin 3 (TXD)
Pin 3 (RTS)	
Pin 4 (CTS)	
Pin 5 (0v)	Pin 4 (0v)

In addition a link must be fitted between Pin 3 (RTS) and Pin 4 (CTS) on the Paradigm unit.