IEC61131-3 PROGRAMMING TRAINING MANUAL

TE 33AU2C2-01

CONTENTS

1 Overview and Common Functions
2 Structured Text
3 Function Block Diagrams
4 Ladder Diagrams
5 Instruction List
6 Sequence Function Charts

Copyright by Yokogawa

This manual is copyright and no part of the document may be copied for uses other than personal study without the written permission of Yokogawa. Furthermore, the manual may not be used for training by non-certified personnel without the written permission Yokogawa.
Training Course Details for
IEC61131-3 Programming

Objectives : The purpose of this course is to familiarize Stardom programmers with the IEC61131 languages used in the Stardom controllers. This course can only be taken in conjunction with the Stardom engineering course.

Course Duration : 2 days

Participants : For engineers involved in software generation or modification of the Stardom system.

Prerequisite : This course is to be taken as part of the Stardom Engineering training course. Some prior experience in programming PLCs would be an advantage.

About the documentation

This manual covers the five languages of the IEC61131-3 standard used in the Stardom controllers. It is divided into six sections:

1. Section 1 – Items common to the programming languages such as data types.
2. Section 2 – Structured Text (ST).
3. Section 3 – Function Block Diagrams (FBD).
4. Section 4 – Ladder Diagrams (LD).
5. Section 5 – Instruction List (IL)
6. Section 6 – Sequence Function Charts (SFC).

Each section gives theoretical detail about the topic, followed by the procedures for carrying out the particular function, where appropriate.

Reference:

This manual was created using the following reference:

Lewis, R.W., *Programming industrial control systems using IEC 1131-3, Revised Edition*  
IEE Control Engineering Series 50, ISBN 0 85296 950 3
# Course Timetable

<table>
<thead>
<tr>
<th>Day</th>
<th>Time</th>
<th>Lectures</th>
<th>ref</th>
<th>Workshops</th>
<th>ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9am</td>
<td>Introduction to IEC61131</td>
<td>1.1-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9.15am</td>
<td>Functions &amp; Function Blocks</td>
<td>1.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9.45am</td>
<td>Data</td>
<td>1.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10.15am</td>
<td>Using Help</td>
<td>1.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10.30am</td>
<td>tea break</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10.45am</td>
<td>Structured Text – Introduction</td>
<td>2.1</td>
<td>Example</td>
<td>2.4</td>
</tr>
<tr>
<td></td>
<td>11.15am</td>
<td>- Concepts</td>
<td>2.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>11.30am</td>
<td>- Statements</td>
<td>2.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>12pm</td>
<td>Lunch</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1pm</td>
<td>FBD – Introduction</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.15pm</td>
<td>- Methods</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3pm</td>
<td>tea break</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3pm</td>
<td>FBD workshop</td>
<td></td>
<td>Example</td>
<td>3.8</td>
</tr>
<tr>
<td></td>
<td>5pm</td>
<td>finish</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>9am</td>
<td>Ladder Diagrams – Concepts</td>
<td>4.1</td>
<td>Example</td>
<td>4.3</td>
</tr>
<tr>
<td></td>
<td>10.30am</td>
<td>tea break</td>
<td>4.2</td>
<td>Example</td>
<td>4.3</td>
</tr>
<tr>
<td></td>
<td>10.45am</td>
<td>Instruction List</td>
<td>5</td>
<td>Example</td>
<td>5.6</td>
</tr>
<tr>
<td></td>
<td>12pm</td>
<td>lunch</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1pm</td>
<td>SFC – Concepts</td>
<td>6.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2pm</td>
<td>- Method</td>
<td>6.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3pm</td>
<td>tea break</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3.15pm</td>
<td>- Execution</td>
<td>6.3</td>
<td>Example</td>
<td>6.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Procedures</td>
<td>6.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5pm</td>
<td>finish</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Revision Log

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>FCX/VDS Revision</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Dec 2001</td>
<td>R0/R4</td>
<td>D. Walker</td>
</tr>
<tr>
<td>1</td>
<td>March 2002</td>
<td>R0/R4</td>
<td>D. Walker</td>
</tr>
<tr>
<td>1.1</td>
<td>April 2002</td>
<td>R0/R4</td>
<td>D. Walker</td>
</tr>
</tbody>
</table>
SECTION 1

STARDOM

SYSTEM OVERVIEW
CONTENTS

1.1 Introduction .......................................................................................................................... 3
1.2 System Configuration .......................................................................................................... 7
  1.2.1 Autonomous Controller Configuration ........................................................................ 8
  1.2.2 Third Party SCADA Systems ...................................................................................... 9
  1.2.3 Integrated Control System ........................................................................................... 10
  1.2.4 System capacity and Specifications ............................................................................ 11
1.3 VDS Overview .................................................................................................................... 12
1.4 Field Controller Hardware Overview ................................................................................ 13
  1.4.1 FCJ .................................................................................................................................. 13
    1.4.1.1 Physical Layout .................................................................................................... 13
    1.4.1.2 CPU module specifications ............................................................................... 13
    1.4.1.3 Communication Specifications .......................................................................... 13
    1.4.1.4 I/O specifications ................................................................................................ 13
  1.4.2 FCN ............................................................................................................................... 14
    1.4.2.1 Physical Layout .................................................................................................... 14
    1.4.2.2 CPU module specifications ............................................................................... 14
    1.4.2.3 Communication Specifications .......................................................................... 15
    1.4.2.4 I/O specifications ................................................................................................ 15
1.5 System Licenses .................................................................................................................. 16
  1.5.1 FCX ............................................................................................................................... 16
  1.5.2 VDS ............................................................................................................................... 17
  1.5.3 Application Portfolio (APPF) Licenses ....................................................................... 18
1.6 Installation .......................................................................................................................... 19
1.1 Introduction

Stardom is a Networked-based Control System (NCS) that is positioned between the lower-end PLC/SCADA systems and the higher-end DCS.

It is a component-based system, in which each component can be used independently of the other components in the system. However, it can also be fully integrated such that all components connect automatically, and the tag database can be globally defined, as with a DCS.
The main features of the system are:

**Control Functions**

- **IEC61131** standard for programming, which specifies five languages as follows:
  - Function Blocks (FB)
  - Ladder Diagrams (LD)
  - Sequence Functions Charts (SFC)
  - Structured Text (ST)
  - Instruction List (IL)

- **Application Portfolios** - user-defined programs that can be security protected for use in other projects. It is based on the use of layering such that function blocks can be created from other function blocks and be in-turn used in other function blocks. Application Portfolios can be created from a set of function blocks to define a control simple or complex control strategy, and can be re-used in other projects.

- **JavaScript** for web and email applications, data processing applications and control functions.

- **Embedded web server** and email function in the field controllers, providing autonomous functionality of the controller. No SCADA HMI (such as the VDS) is required, making them ideal for embedded and remote applications.

- **Full redundancy** in controller hardware. Power Supply, CPU and Network connections can all be made redundant.

**Network Functions**

- **HSE** (high speed ethernet) communications between controllers and data server.
- **OPC** client and server communications between the Data Server and other devices (host computers, PLCs, etc).
- **Standard Web interface** to operator displays, i.e., the operator stations are PCs with IE5, and no proprietary software is required.
- **Field Networking** – the controllers communicate to field transmitters, PLCs, and other devices over Foundation Fieldbus, Profibus, Modbus and a range of other field networks. (Note: Profibus and Modbus are not yet implemented).

- **Redundancy** – full network redundancy is available between the controllers and the VDS, i.e. the control network.

**Data Server Functions**

- The VDS can communicate to third party devices, as well as to the FCJ and FCN (hereafter referred to collectively as FCX). This provides integration of the control hardware in a site to a central point. This means that a DCS type global database ensures easier tag management. See below for more information on system configuration.
A short description of each Stardom component is as follows:

**FCJ – Field Control Junction**

A small autonomous field controller with a small amount of fixed analog and digital I/O and communication ports for remote I/O, PLCs and Fieldbus. Programmable with IEC61131 standard programming languages and Java.

**FCN – Field Control Node**

A rack mounted unit with up to three racks of I/O modules. I/O modules also include communication modules for remote I/O, PLCs and Fieldbus. It is programmable with IEC61131 standard programming languages and Java, and has the same CPU as the FCJ.

**VDS – Versatile Data Server**

A PC based application that combines a data server and a web server to collect data from FCX’s, FA-M3 PLCs, DAQ Stations and third party devices and provide information to the user via web-based operator displays. The data server and web server connect to each other using OPC, and can be in the same or separate PCs. Third party applications can also access the data server using OPC.

**Engineering Tools**

A suite of engineering tools are used to program the field controllers and the VDS. This includes:

- **Logic Designer** - IEC61131 programming interface
- **Resource Configurator** – hardware configuration tool
- **VDS Builder (Object Builder, Graphic Designer)** – a set of tools for building the VDS, including a graphics development package.
Software Structure of the System:

The diagram below shows the functions and applications within the FCX and VDS, and the builder applications that are used to configure these functions.
1.2 System Configuration

A Network Based Control system can be set up in many different configurations, using all or part of the system.
1.2.1 Autonomous Controller Configuration

- The FCX, can operate as standalone devices without an HMI station connected to it.

- The FCX has an in-built web server, so that it can provide its own operator interface. These displays are built with a standard html editor, not the VDS graphics builder.

- The FCX has a serial port to connect to a video display panel. See the General Specification (GS 34P02Q01) for details on display panels that can be currently connected to the FCX.
1.2.2 Third Party SCADA Systems

- The FCX can connect to other SCADA HMI’s. These drivers are currently being developed. See the Yokogawa Stardom website for the latest drivers available.

- It is not necessary for a VDS to be connected.
1.2.3 Integrated Control System

- The VDS can connect to FCX’s to create an integrated control system with global database. In this way, the system is similar to a DCS.

- The VDS can communicate to other devices, such as the Yokogawa PLC (FA-M3), the Yokogawa data acquisition unit (DAQ Station), and third party PLCs, on ethernet. The VDS can use specially written drivers for these devices, or use standard OPC.

- Other SCADA systems can reside within the same system.
1.2.4 System capacity and Specifications

**Capacity:**

- Number of FCX per VDS: 32 per Data Server
- Number of VDS to a system: 4 Data Servers per Web Server (*1)
- Number of HMI Clients: 50 Clients per Web Server
- Number of Web Servers: 1 per system
- Number of objects (tags) per VDS: 5,000 per Data Server
- Number of FCXs communications: 15 FCXs can connect to each other

(*1) A VDS comprises a Data Server and a Web Server. These can be in separate PCs and several data servers can serve one web server. A maximum of 4 data servers can connect to 1 web server. There can be only 1 web (HMI) server per system.

**Specifications:**

- Control Network: Standard Ethernet (10/100 Mbps) Dual Redundancy available
- Connection protocol between FCX and VDS: HSE (client/server implementation)
- Connection protocol between Data Server and Web Server: OPC
1.3 VDS Overview

The VDS software resides in a PC. Because the VDS comprises two main functions, Data Server and Web Server, these can reside in separate PCs with an OPC connection between them. The required PC specifications for the VDS software is as follows:

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Windows 2000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Service Pack 2</td>
</tr>
<tr>
<td>CPU</td>
<td>P3-700 Mhz</td>
</tr>
<tr>
<td>Memory</td>
<td>256MB</td>
</tr>
</tbody>
</table>

*The VDS function may be shown schematically as follows:*
1.4 Field Controller Hardware Overview

1.4.1 FCJ

1.4.1.1 Physical Layout

The FCJ is a single encapsulated unit with fixed I/O and Foundation Fieldbus ports. It also has a PC-MCIA port for Flash RAM, fixed and portable memory for the controller.

1.4.1.2 CPU module specifications

The CPU is a Pentium MMX 166 MHz. It does not require a fan for cooling.

<table>
<thead>
<tr>
<th>CPU</th>
<th>Pentium MMX 166</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>32 MB main memory with ECC</td>
</tr>
<tr>
<td></td>
<td>512 KB static RAM with ECC</td>
</tr>
<tr>
<td></td>
<td>Slot for Flash-RAM card (32MB)</td>
</tr>
<tr>
<td>Fan</td>
<td>Not required</td>
</tr>
</tbody>
</table>

1.4.1.3 Communication Specifications

The FCJ has the following communications ports:

<table>
<thead>
<tr>
<th>Type</th>
<th>Ports</th>
<th>Speed</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet</td>
<td>2</td>
<td>100/10 Mbps</td>
<td>dual-redundant communications, or for independent communications to PLCs and other devices</td>
</tr>
<tr>
<td>Serial</td>
<td>2</td>
<td>115.2 Kbps</td>
<td>for local HMI interface, programming, or PLC interface</td>
</tr>
</tbody>
</table>

1.4.1.4 I/O specifications

<table>
<thead>
<tr>
<th>Type</th>
<th>Qty</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analog Inputs</td>
<td>6</td>
<td>1 – 5 V DC, non-isolated</td>
</tr>
<tr>
<td>Outputs</td>
<td>2</td>
<td>4-20 mA dc, non-isolated</td>
</tr>
<tr>
<td>Digital* Inputs</td>
<td>16</td>
<td>rating: 4.1mA @ 24 VDC</td>
</tr>
<tr>
<td>Outputs</td>
<td>16</td>
<td>rating: 100mA @ 24 VDC</td>
</tr>
<tr>
<td>Communications</td>
<td>2</td>
<td>Foundation Fieldbus port</td>
</tr>
</tbody>
</table>

*Digital I/O is point-to-point non-isolated, but field-to-circuit isolated.
1.4.2 FCN

1.4.2.1 Physical Layout

The FCN is a rack-mounted controller with up to three racks, each with 10 slots for CPU and I/O modules, and 2 separate slots for the Power Supplies.

The SB I/F card provides a high-speed data connection between the I/O racks. If there are no expansion racks, then a total of 8 I/O modules can be installed in the rack, as no SB I/F module is required.

Note that the PSU, CPU and SB I/F modules can be dual redundant:

- In the case of the CPU module, this takes two more I/O slots.
- In the case of the SB I/F module, it takes one more I/O slot.

1.4.2.2 CPU module specifications

The CPU is a Pentium MMX 166 MHz. It does not require a fan for cooling.

<table>
<thead>
<tr>
<th>CPU</th>
<th>Pentium MMX 166</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>32 MB main memory with ECC</td>
</tr>
<tr>
<td></td>
<td>512 KB static RAM with ECC</td>
</tr>
<tr>
<td></td>
<td>Slot for Flash-RAM card (32 MB)</td>
</tr>
<tr>
<td>Fan</td>
<td>Not required</td>
</tr>
</tbody>
</table>
1.4.2.3 Communication Specifications

The FCN has communications ports on the CPU card, plus plug-in communications cards in the I/O rack. The specification for the CPU communications ports are as follows:

<table>
<thead>
<tr>
<th>Type</th>
<th>Ports</th>
<th>Speed</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet</td>
<td>2</td>
<td>100/10 mbps</td>
<td>dual-redundant communications, or for independent communications to PLCs and other devices</td>
</tr>
<tr>
<td>Serial</td>
<td>1</td>
<td>115.2 kbps</td>
<td>for local HMI interface, programming, or PLC interface</td>
</tr>
</tbody>
</table>

1.4.2.4 I/O specifications

A large range of I/O and communication modules can be installed in the FCN. Up to 25 modules may be installed in an FCN that does not have dual-redundant CPU or SB interface, or 20 modules in a fully redundant system.

A summary of I/O modules is as follows. Full specification are available in the General Specification sheet: 34P02Q31-01E.

<table>
<thead>
<tr>
<th>Type</th>
<th>Specification</th>
<th>Isolation</th>
<th>Channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analog Input</td>
<td>Current (4-20mA)</td>
<td>non-isolated</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>isolated</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Voltage (1-5 Vdc)</td>
<td>non-isolated</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>Voltage (-10-+10Vdc)</td>
<td>non-isolated</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>Thermocouple/mV</td>
<td>isolated</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>RTD</td>
<td>isolated</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>Pulse</td>
<td>isolated</td>
<td>8</td>
</tr>
<tr>
<td>Analog Output</td>
<td>Voltage</td>
<td>non-isolated</td>
<td>16</td>
</tr>
<tr>
<td>Analog Input/Output</td>
<td>Voltage in/Current out</td>
<td>non-isolated</td>
<td>8 in/8 out</td>
</tr>
<tr>
<td></td>
<td>Current</td>
<td>non-isolated</td>
<td>4 in/4 out</td>
</tr>
<tr>
<td>Digital Input</td>
<td>24 VDC</td>
<td>note 1</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>100 VAC</td>
<td>note 1</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>200VAC</td>
<td>note 1</td>
<td>16</td>
</tr>
<tr>
<td>Digital Output</td>
<td>Transistor</td>
<td>note 1</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Relay</td>
<td>note 1</td>
<td>16</td>
</tr>
</tbody>
</table>

Note 1: Digital I/O is point-to-point non-isolated, but field-to-circuit isolated.
1.5 System Licenses

1.5.1 FCX

The model number of the FCJ is built as follows:

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFJT100</td>
<td>FCJ autonomous controller</td>
</tr>
<tr>
<td>-S</td>
<td>Without field network interface</td>
</tr>
<tr>
<td>1</td>
<td>With analog and digital I/O (6 AIs, 2 AOs, 16 DIs, 16 DOs)</td>
</tr>
<tr>
<td>0</td>
<td>Always 0</td>
</tr>
<tr>
<td>0</td>
<td>General-purpose model</td>
</tr>
</tbody>
</table>

The model number of the FCN is built as follows:

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NFBU200-S00</td>
<td>Base Module (back plane)</td>
</tr>
<tr>
<td>NFPW441/2/4-10</td>
<td>Power Supply Module (110/240VAC/24VDC)</td>
</tr>
<tr>
<td>NFCP100-S00</td>
<td>CPU Module</td>
</tr>
<tr>
<td>NFSB100-S00</td>
<td>SB Bus Repeater</td>
</tr>
</tbody>
</table>

See GS 34P02Q12-01E for more information on FCN model numbers.

Licensing of the FCXs consists of the following:

- FCN/FCJ Basic Software License
- I/O Credit License

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NT203AE-PC11A</td>
<td>FCN/FCJ Software Medium</td>
</tr>
<tr>
<td>NT711AE-LM01A</td>
<td>FCN/FCJ Basic Software License (single CPU)</td>
</tr>
<tr>
<td>NT712AE-LM01A</td>
<td>FCN/FCJ Basic Software License (dual CPU)</td>
</tr>
<tr>
<td>NT720AE-xxxx</td>
<td>Additional I/O Credit License</td>
</tr>
</tbody>
</table>

Example:

For a system with:

20 AI, 10 AO, 16 DI, 16 DO

I/O Credits = 5x20 + 10x10 + 16x1 + 16x1 = 232

The license model number is: NT720AE-0232
1.5.2 VDS

The VDS licensing consists of basic software licenses for the VDS and for the HMI client. In addition, there are licenses for the development version (Full-time) or the runtime version. Each PC running the VDS (or one of its components) requires an ID module (which can be USB or printer port type).

*The license model numbers are as follows:*

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NT201AE-PC11A</td>
<td>VDS Software Media</td>
</tr>
<tr>
<td>NT610FE-LU/Pxx</td>
<td>VDS Basic Software License (Full-time) (USB/Parallel)</td>
</tr>
<tr>
<td>NT610RE-LU/Pxx</td>
<td>VDS Basic Software License (Run-time) (USB/Parallel)</td>
</tr>
<tr>
<td>NT625FE-LWyyA</td>
<td>HMI Client Additional Licenses</td>
</tr>
</tbody>
</table>

The Basic Software License is based on the number of Control Objects, that is, the number of objects being read from the FCXs or other controllers connected to the VDS. The ‘xx’ is built as follows:

<table>
<thead>
<tr>
<th>xx</th>
<th>Control Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>50 Objects</td>
</tr>
<tr>
<td>01</td>
<td>100 Objects</td>
</tr>
<tr>
<td>02</td>
<td>200 Objects</td>
</tr>
<tr>
<td>05</td>
<td>500 Objects</td>
</tr>
<tr>
<td>10</td>
<td>1000 Objects</td>
</tr>
<tr>
<td>15</td>
<td>1500 Objects</td>
</tr>
<tr>
<td>20</td>
<td>2000 Objects</td>
</tr>
<tr>
<td>50</td>
<td>5000 Objects</td>
</tr>
</tbody>
</table>

The basic license provides for 5 HMI clients. For additional licenses, the HMI Client Additional License ‘yy’ value is built as follows:

<table>
<thead>
<tr>
<th>yy</th>
<th>No. of Clients</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Add 1 Client</td>
</tr>
<tr>
<td>05</td>
<td>Add 5 Clients</td>
</tr>
</tbody>
</table>

In addition to the basic licenses, the following optional packages are also available:

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NT301AE/RE</td>
<td>Report Package</td>
</tr>
<tr>
<td>NT302AE/RE</td>
<td>Trend Package</td>
</tr>
<tr>
<td>NT351AE</td>
<td>Melsec Driver</td>
</tr>
<tr>
<td>NT356AE</td>
<td>SYSMAC Driver</td>
</tr>
<tr>
<td>NT358AE</td>
<td>OPC Server Driver</td>
</tr>
<tr>
<td>NT365AE</td>
<td>DARWIN Driver</td>
</tr>
<tr>
<td>NT366AE</td>
<td>Power Monitor Driver</td>
</tr>
<tr>
<td>NT303AE/RE</td>
<td>Test Package</td>
</tr>
<tr>
<td>NT304AE/RE</td>
<td>Multi-Task Package</td>
</tr>
</tbody>
</table>
1.5.3 Application Portfolio (APPF) Licenses

As well as each FCX requiring a license, a license is also required for each Application Portfolio for each FCX. There are a range of standard APPF licenses available created by Yokogawa, such as:

<table>
<thead>
<tr>
<th>APPF</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAS Portfolio</td>
<td>Yokogawa function blocks</td>
</tr>
<tr>
<td>SAMA Portfolio</td>
<td>function blocks for the power generation industry employing the SAMA standard</td>
</tr>
<tr>
<td>General Use Portfolio</td>
<td>General purpose functions developed by Yokogawa such as cross-limiting, resource scheduler and batch counter.</td>
</tr>
<tr>
<td>Communication Portfolio</td>
<td>Communication drivers for PLCs.</td>
</tr>
<tr>
<td>AP Specific Portfolio</td>
<td>High level application blocks developed by Yokogawa, such as drum level correction.</td>
</tr>
</tbody>
</table>

A license must be purchased for each APPF for each FCX.

For example:

<table>
<thead>
<tr>
<th>FCX1</th>
<th>PASPOU + SAMAPOU</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCX2</td>
<td>PASPOU</td>
</tr>
</tbody>
</table>

3 APPF licenses must be purchased.

In addition to the standard portfolios produced by Yokogawa, users can develop their own as described in section 2.1.4. These can then become APPFs that require licenses by other users to install.
1.6 Installation

The following software needs to be installed to use the system:

- FCN/FCJ Installation CD:
  - Resource Configurator
  - Logic Designer

- APPF CD:
  - Application Portfolios

- VDS CD:
  - VDS Software
  - Java Plug-in

Pre-requisites for installing the software:

- Windows 2000, Service Pack 2
- Internet Explorer 5.5 or greater

Procedure for installing the software:

1. Create a user account in Windows called ‘stardom’
2. This account must have Administrators authority
3. Logon as Administrator
4. Insert the FCX/FCJ CD
   - The installation page will automatically appear.
   - Click on Resource Configurator and select Install
   - When complete, click on Logic Designer and select Install
   - It is not necessary to install the PAS-POU libraries as they are automatically installed when Logic Designer is installed (if selected).
   - Reboot the computer
   - See Section 2.2.5.3 for information on setting the licenses for the FCX.
5. Logon as Administrator
6. Insert the APPF CD (optional)
   - The installation page will automatically appear.
   - Click on Install Software and select the required APPF
   - Repeat this procedure for each APPF that is required.
   - There are examples available on the CD that can be copied over and used for reference.
7. Logon as Administrator
8. Insert the VDS CD
   - The installation page will automatically appear.
   - Click on Install Software and select VDS
   - After installing the VDS software, repeat the procedure to install Java Plug-in
   - Reboot the PC

Setting up VDS to run:

1. Logon as the user account for VDS (stardom) and register licenses with “license management tool” (see section 3.2.3).
2. Perform Initial Setup with the Initial Setup tool. A reboot is required.

3. VDS will now start whenever the VDS account (stardom) is logged on.
SECTION 1

IEC61131-3 PROGRAMMING

Overview and Common Functions
CONTENTS

1.1 INTRODUCTION ............................................................................................................................................... 3

1.2 OVERVIEW OF THE IEC61131-3 STANDARD .................................................................................................. 4

1.3 FUNCTIONS AND FUNCTION BLOCKS ........................................................................................................... 6
  1.3.1 COMMON ELEMENTS ............................................................................................................................... 6
  1.3.2 FUNCTIONS ............................................................................................................................................... 8
  1.3.3 FUNCTION BLOCKS ..................................................................................................................................... 9

1.4 DATA ............................................................................................................................................................. 10
  1.4.1 VARIABLE TYPES ..................................................................................................................................... 10
  1.4.2 DECLARING VARIABLES ........................................................................................................................... 12
  1.4.3 DATA TYPES ............................................................................................................................................... 15
  1.4.4 CREATING DATA TYPES ........................................................................................................................... 18

1.5 USING HELP IN LOGIC DESIGNER .................................................................................................................. 25
1.1 Introduction

This course covers the programming languages utilized by the Stardom controllers. The programming conforms to the IEC61131-3 standard which defines a complete programming system, data structure and PLC interface.

IEC61131-3 is part of IEC61131 standard, which deals with all aspects of PLC application to industrial-process measurement and control.

IEC61131-3 is a PLC language standard defines the following programming methods:

- Function Block Diagrams
- Ladder Diagrams
- Instruction list
- Structured Text
- Sequence Function Charts

The standard was developed to standardize the programming of PLCs across the many hardware platforms available on the market. This has several advantages:

- A common language means that users do not have to learn different languages for different PLCs.
- Portability of programs across different PLCs.
- Well structured ‘top-down’ or ‘bottom-up’ program development.
- Allows a program to be broken down into functional elements called Program Organization Units (POUs). POUs include functions, function blocks and programs and can be developed as portfolios by users for re-use.
- Strong Data typing - Various data elements that are associated can be addressed as a single entity
- Self check to detect when a wrong data type is being assigned to a variable.
- Full execution control of scan time – facilities are provided for different parts of a program to be executed at:
  - different times
  - different rates
  - in parallel
- Can describe Complex Sequential Behavior
1.2 Overview of the IEC61131-3 Standard

Configurations

Configurations are at the highest level of the project. They define the hardware interface to the PLC, containing the configuration to enable interfacing between the software and the hardware.

This includes

- PLC specification (resource)
- Tasks
- I/O Configuration
- Global Variables

Resources

Within each configuration there are one or more resources. A resource provides system support for all of the features needed for the execution of programs and an interface between a program and the physical I/O channels of the PLC.

An IEC program cannot function unless it is loaded into a resource

IEC standard allows each resource to be able to support more than one program

Program Organisation Units (POUs)

An IEC program can be built from a number of different software elements. Each software element can be written in any one of the different IEC languages. Typically a program consists of a number of interconnected Function Blocks.

The execution of a program is controlled using tasks. A POU must be instantiated into a task to execute.

Tasks

Controls a set of programs and/or FBs to execute periodically or upon the occurrence of a specified trigger.

A program or FB will remain dormant unless it is assigned to a specific task and the task is configured to execute periodically or when triggered by a specified variable.
**Function Blocks (FBs)**

One of the most important building block within IEC 1131-3 for creating programs.

Any FB can be programmed using other FBs. This helps to create programs that are well structured and truly hierarchical.

**Functions**

A software element which when executed, with a particular set of input values produces one primary result.

Unlike a Function Block, it has no internal storage.

Examples:

SIN(), COS() functions

**Control flow**

Configurations and resources are responsible for the control and execution of elements in them. Configurations and resources can be started and stopped by external facilities such as Operator Interface or HIM.

- **Start-up (defined in standard)**
  - Configuration started, all global variables initialized and all resources are then started
  - When a resource is started, all variables within the resource are initialized and all tasks are then enabled
  - Once tasks are enabled, all programs and FBs assigned to the tasks will start to execute
  - Starting a configuration results in starting all software elements that exist within it

- **Shut-down (defined in standard)**
  - When a configuration is stopped, all contained resources are stopped
  - When a resource is stopped, all tasks are disabled and programs and FBs cease to execute
1.3 Functions and Function Blocks

1.3.1 Common elements

**Identifiers**

Identifiers are used for naming different elements, such as:

- Variables,
- new data types,
- FBs
- Programs (POUs)

An identifier can be any string of letters, digits and underlines provided that:

- The first character is not a digit and
- There are not two or more underline characters together
- There are no embedded spaces

Lower case letters and identifiers with a leading underline character are supported.

**Acceptable identifiers:**

W123_PV, W12_3PV, aTemp1, _PROG1

**Unacceptable identifiers:**

W123__PV, W12 3PV, 1Tempa, Q%TY12

The Standard states that the first six characters should be tested for uniqueness:

Hence A12345_XY and A12345_GG may be regarded as identical on some systems

To avoid ambiguous identifiers, ensure the first six characters are unique.
Comments

Various length comments from short to multi-line can be inserted in ST, FDBs, LD, SFCs. There are some restrictions in IL.

They can be placed wherever it is acceptable to insert one or more spaces.

Comments cannot be nested.

Comment usage:

Framed by the characters (*….*)

Example:

(* Activate Pump*)
(************************************************)
(* Main turbine interlock logic *)
(************************************************)
1.3.2 Functions

Functions are POUs with the following attributes:

- They have multiple input parameters and exactly one output parameter.
- They have no internal memory (i.e. no internal variables for storing data).
- Calling a function with the same values returns always the same result.
- Return values can be single data types.
- Within a function it is possible to call another function but not a function block or a program. Recursive calls are not allowed.
- The abbreviation for functions is FU.
- A Function is not given an identifier when it is placed on a worksheet.

The following is a list of standard IEC 61131-3 function types:

- Type conversion functions, such as INT_TO_REAL
- Numerical functions, such as ABS and LOG
- Standard arithmetic functions, such as ADD and MUL
- Bit-string functions, such as AND and SHL
- Selection and comparison functions, such as SEL and GE
- Character string functions, such as RIGHT and INSERT
- Functions of time data types, such as SUB with the data type TIME

In Logic Designer, EDIT WIZARD, these are grouped as follows:

<table>
<thead>
<tr>
<th>Group</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functions</td>
<td>Numerical</td>
<td>ABS, LOG, SIN</td>
</tr>
<tr>
<td></td>
<td>Arithmetic</td>
<td>ADD, DIV, MUL, SUB</td>
</tr>
<tr>
<td></td>
<td>Bit-string</td>
<td>AND, OR, NOT</td>
</tr>
<tr>
<td></td>
<td>Selection and Comparison</td>
<td>GT, GE, LT, LE</td>
</tr>
<tr>
<td></td>
<td>Time data types</td>
<td>ADD_T_T, DIV_T_AI</td>
</tr>
<tr>
<td>String FUs</td>
<td>Character string functions</td>
<td>RIGHT, INSERT</td>
</tr>
<tr>
<td>Type conv. FUs</td>
<td>Type Conversion</td>
<td>INT_TO_REAL</td>
</tr>
</tbody>
</table>

In addition, there are a large range of other Functions created by Yokogawa for control related functions specific to Yokogawa. Details of these blocks are covered in the Stardom Engineering course.

Details about each function are available in the Logic Designer HELP. To view details of a Function:

1. In EDIT WIZARD, right-click on the function.
2. Select ‘Help on FB/FU’.
3. The HELP page will open with the details of the Function.
1.3.3 Function Blocks

Function blocks are POU.s with the following attributes:

- Multiple input/output parameters and internal memory.
- The value that a function block returns depends on the value of its internal memory.
- Within a function block it is possible to call another function block or functions but not a program.
- Recursive calls are not allowed.
- The abbreviation for function blocks is FB.

The following is a list of standard IEC 61131-3 function blocks:

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bistable elements</td>
<td>RS</td>
<td>Reset Dominant Latch</td>
</tr>
<tr>
<td></td>
<td>SR</td>
<td>Set Dominant Latch</td>
</tr>
<tr>
<td>Edge detection</td>
<td>R_TRIG</td>
<td>Rising Edge Detection</td>
</tr>
<tr>
<td></td>
<td>F_TRIG</td>
<td>Falling Edge Detection</td>
</tr>
<tr>
<td>Counters</td>
<td>CTU</td>
<td>Counter Up</td>
</tr>
<tr>
<td></td>
<td>CTD</td>
<td>Counter Down</td>
</tr>
<tr>
<td></td>
<td>CTUD</td>
<td>Counter Up/Down</td>
</tr>
<tr>
<td>Timer functions</td>
<td>TOF</td>
<td>Timer Off-Delay</td>
</tr>
<tr>
<td></td>
<td>TON</td>
<td>Timer On-Delay</td>
</tr>
<tr>
<td></td>
<td>TP</td>
<td>Pulse</td>
</tr>
</tbody>
</table>

In addition to these, Yokogawa have created a large range of function blocks for the Stardom controller, including CS3000 type function blocks. Details of these blocks are covered in the Stardom Engineering course.

Details about each function block are available in the Logic Designer HELP.
1.4 Data

Note: for detailed information about data and data types, go to the Help files in Logic Designer. Select CONTENTS – IEC61131 – “Data types in IEC61131”.

1.4.1 Variable Types

In IEC61131, every function and function block contains variables, these can be inputs and outputs to function or function block, or (in the case of function blocks only) can be internal or local variables.

In the case of a PAS_PID block it has 12 Input Variables and 4 Output Variables. It may also have variables connected to some of these inputs/outputs. In addition there are internal variables (such as PID and alarm parameters) that exist inside it.

Data is in the form of global variables or local variables:

- **Local Variables** – internal data that is used within a POU (function block or program) and not required to be accessed elsewhere. This can be in the form of a Variable, or Input/Output Variable.

- **Global Variables** – data that is available to all POUs in the FCX. It is declared as an External Variable within a POU, and as a Global Variable in the Resource (FCX) folder.

Data within function blocks are assigned as follows:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAR</td>
<td>Local Variable</td>
<td>Internal Data</td>
</tr>
<tr>
<td>VAR_EXTERNAL</td>
<td>Global Variable</td>
<td>Data to be used in other blocks</td>
</tr>
<tr>
<td>VAR_IN_OUT</td>
<td>An Input/Output Variable</td>
<td>Data that is an input to and output from the function block.</td>
</tr>
<tr>
<td>VAR_INPUT</td>
<td>An Input Variable</td>
<td>Data that is an input to the function block.</td>
</tr>
<tr>
<td>VAR_OUTPUT</td>
<td>An Output Variable</td>
<td>Data that is an output from the function block.</td>
</tr>
</tbody>
</table>

If a variable is used only within a POU it is called local variable. In those cases the variable keywords VAR, VAR_INPUT and VAR_OUTPUT are used.

If a variable is be used within the whole project it is called a global variable. It has to be declared as VAR_GLOBAL in the global declaration and as VAR_EXTERNAL in each POU where it is used.
The relationship between local and global variables, and the FCX software configuration is as follows:
1.4.2 Declaring Variables

Data must be declared to be used by the system. The Logic Designer declares most data automatically as it is created.

When data is declared, its **data type** is specified. There are a large set of standard data types in IEC61131, and these are listed below. A new data type can be made up of these data types by declaring them as **structures**, **arrays** or **strings**, in the **Data Types** folder. Data can be registered in the following ways:

- **Every POU** that is created includes a data folder in which all local and global variables related to that block are registered. This happens automatically as the function blocks are created. The data type is set when the variable is created, and can be changed at any time in the Variable file of that POU.

- **Global variables** are created in the FCX itself in the Physical Hardware folder. If a variable in a POU is declared as a Global Variable, then it will appear automatically in the Global Variable folder. However, a variable could be declared first in the Global Variable folder, and then used as many times as needed in the POUs. The data type is set when the variable is created and can be changed in the Global Variable folder.

- **I/O** are created in the DeviceLabelDefinition file, and are automatically registered in the Global Variable folder in the relevant ‘DeviceLabel’ section at the bottom of the global variable list. When assigned in the POU, it will be declared as a VAR_EXTERNAL.

If a variable is used only within a POU it is called **local variable**. In those cases the variable keywords VAR, VAR_INPUT and VAR_OUTPUT are used.

If a variable is be used within the whole project it is called a **global variable**. It has to be declared as VAR_GLOBAL in the global declaration and as VAR_EXTERNAL in each POU where it is used.
How to Register Data:

1. Double click on a connection point on a function block or logic element of a ladder.
2. Type in the name of the variable, and select either Local and Global. See next page for more information on selecting a Global Variable.
3. In the COMMON tab, select Variable Type (Usage) and Data Type. If the data is to be read by another system, such as the VDS for graphics, then select OPC.
How to Register Global Variable Data:

1. A Global Variable must first be declared in the Global Variable Definition file in the FCX as follows:

2. This Global Variable can now be used in the POUs.
3. In a POU, double click on a connection point on a function block or logic element of a ladder.
4. Click on the Global Scope tab and select the Resource (FCX) and the category of Global Variable in which the variable is to be placed, in this case, User1.
5. Click on the Contact or Variables tab and type in the name of the variable, and select Global.
6. Everything is grayed out in Common, and OPC cannot be set. This is set in the Global Variables page in the FCX folder.
1.4.3 Data Types

Every variable has an attribute known as a data type, and each variable must be declared with this data type. There are several categories of data types as follows:

**Integers**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
<th>Bits</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>SINT</td>
<td>Short integer</td>
<td>8</td>
<td>-128 to +127</td>
</tr>
<tr>
<td>INT</td>
<td>Integer</td>
<td>16</td>
<td>-32768 to +32767</td>
</tr>
<tr>
<td>DINT</td>
<td>Double integer</td>
<td>32</td>
<td>-2³¹ to +2³¹-1</td>
</tr>
<tr>
<td>USINT</td>
<td>Unsigned short integer</td>
<td>8</td>
<td>0 to 255</td>
</tr>
<tr>
<td>UINT</td>
<td>Unsigned integer</td>
<td>16</td>
<td>0 to +2¹⁶-1</td>
</tr>
<tr>
<td>UDINT</td>
<td>Unsigned double integer</td>
<td>32</td>
<td>0 to +2³²-1</td>
</tr>
</tbody>
</table>

**Integer Literals**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>2#</td>
<td>Binary</td>
<td>2#1101_0101</td>
</tr>
<tr>
<td>8#</td>
<td>Octal</td>
<td>8#377</td>
</tr>
<tr>
<td>16#</td>
<td>Hexadecimal</td>
<td>16#FF</td>
</tr>
</tbody>
</table>

Note that the default is INT (16bit) if no designation is used.

**Real Numbers**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
<th>Bits</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL</td>
<td>Real numbers</td>
<td>32</td>
<td>±10³³⁸</td>
</tr>
</tbody>
</table>

**Time and Date**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
<th>Literals</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME</td>
<td>Time duration</td>
<td>T# or TIME# (d,h,m,s,ms)</td>
</tr>
</tbody>
</table>

**Examples of Time and Date literals:**

```
TIME
T#16d3h2m - 16 days, 3 hours, 2 minutes
T#10.12s  - 10.12 seconds
```
### Strings

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
<th>Bits</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRING</td>
<td>Character strings</td>
<td>n/a</td>
<td>‘string’</td>
</tr>
</tbody>
</table>

**Embedded control characters within strings:**

<table>
<thead>
<tr>
<th>Control Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$$</td>
<td>Single dollar sign</td>
</tr>
<tr>
<td>$’</td>
<td>Single quote character</td>
</tr>
<tr>
<td>$L or $l</td>
<td>A line feed character</td>
</tr>
<tr>
<td>$N or $n</td>
<td>A new line character</td>
</tr>
<tr>
<td>$P or $p</td>
<td>Form feed, new page</td>
</tr>
<tr>
<td>$R or $r</td>
<td>Carriage return character</td>
</tr>
<tr>
<td>$T or $t</td>
<td>Tabulation, i.e. tab character</td>
</tr>
</tbody>
</table>

### Bit Strings

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
<th>Bits</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Bit string of 1 bit</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>BYTE</td>
<td>Bit string of 8 bit</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WORD</td>
<td>Bit string of 16 bit</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>DWORD</td>
<td>Bit string of 32 bit</td>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>

### Boolean

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
<th>Bits</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>switch</td>
<td>1</td>
<td>TRUE/FALSE or 1/0</td>
</tr>
</tbody>
</table>
Generic Data

Starts with the prefix ANY_ and is used for describing variables in functions and FBs, where there is support for **overloaded** I/O.

**Overloaded** refers to the ability of a variable to be used for different types of data. The ANY_ variable defines a category of data, a subset of which can be used for the inputs and outputs to that function or FB.

<table>
<thead>
<tr>
<th>Overload</th>
<th>Generic Data Type</th>
<th>Supported Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANY_</td>
<td>ANY_NUM</td>
<td>ANY_REAL, REAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ANY_INT, SINT, INT, DINT, USINT, UINT, UDINT</td>
</tr>
<tr>
<td>ANY_BIT</td>
<td></td>
<td>1.WORD, DWORD, WORD, BYTE, BOOL</td>
</tr>
<tr>
<td>ANY_STRING</td>
<td></td>
<td>STRING</td>
</tr>
<tr>
<td>ANY_DATE</td>
<td></td>
<td>TIME</td>
</tr>
</tbody>
</table>

**Example – the OR Function I/O variables:**
1.4.4 Creating Data Types

When a variable is created it is automatically declared by the system within the data sheet of a Function Block or Program, as one of the data types mentioned above. However, it is also possible to create a data type with which to declare variables. These include composite or structured data types, which allow the encapsulation of different data types into a single entity.

**Data types** are declared under the following categories:

- STRUCTURE
- STRING
- ARRAY
- ENUMERATED DATA

The format for declaring a data type is as follows:

```
TYPE
    Name of data type : (STRUCTURE/STRING/ARRAY)
    Name1 : data type ;
    .    .
    .    .
END_TYPE
```

If the data type is a STRUCTURE, then there will be a list of data under the name of the data type. If the data type is a STRING, ARRAY, or ENUMERATED there can be multiple declarations within the TYPE/END_TYPE field.
Declaring arrays

Array data types include several elements of the same data type. The particular elements within one array are identified and accessed using an array index. An array can be used to declare several elements of the same type with only one line in the type declaration.

Example for the declaration of an array data type:

```plaintext
TYPE
  graph : ARRAY[1..23] OF INT;
END_TYPE
```

In the example the data type 'graph' contains 23 elements of the data type 'INT'. All elements of an array are stored continuously one after another in the PLC memory.

In order to declare an array, you first have to declare the array in the type declaration of the data type worksheet. Subsequently you can declare a variable in the variable worksheet of the POU (refer to the following programming example).

Multi-dimensional arrays

If multi-dimensional arrays are needed, arrays of arrays can be used. An example for an array of an array is shown in the following figure:

```plaintext
TYPE
  graph : ARRAY [1..10] OF INT;
  my_array : ARRAY [1..3] OF graph;
END_TYPE
```

In an array of an array a single element is accessed using two indexes as shown in the following figure:

Variable declaration:

```plaintext
VAR
  var1 : my_array;
  var2 : INT;
END_VAR
```

Code body declaration in ST:

```plaintext
var2 := var1[1][3];
```
Declaring Strings

User defined string data types are strings with a variable number of characters. Declaring a user defined string the length is set in parentheses behind the data type.

*Example for the declaration of an user defined string data type.*

```plaintext
TYPE
    STRING10 : STRING(10)
END_TYPE
```

In this example the string length is 10 characters.

**Limits:**

- the shortest possible string length = 1
- the longest possible string length = 1024

Declaring Enumerated Data

Enumerated data is a data type with a fixed set of values in it.

*Example for the declaration of an enumerated data type.*

```plaintext
TYPE
    VALVE_MODE : (OPEN, SHUT, FAULT) ;
    PUMP_MODE : (RUNNING, OFF, FAULT) ;
END_TYPE
```

In the example the data type 'VALVE_MODE' can have only the values 'OPEN', 'SHUT' or 'FAULT', etc.

These can then be referenced in a program as follows:

```plaintext
IF AX100 = PUMP_MODE#FAULT THEN
    XV23 = VALVE_MODE#OPEN;
```
**Declaring Data Structures**

**Data Structures** are a means of building a series of data of given **data types** into a single object. Thus, a function block will contain a series of data items that are defined as part of the data structure for the block. For example, the PAS_POU block, PAS_PVI, has the following data structure defined:

```plaintext
TYPE (*PVI*)
    SD_PPARAM_PVI : STRUCT
        MODE : DWORD;
        ALRM : DWORD;
        AF : DWORD;
        AOF : BOOL;
        PV : CData_REAL;
        PVCAL : BOOL;
        RAW : CData_REAL;
        SUM : SD_PSUM_DEF;
        HH : REAL;
        LL : REAL;
        PH : REAL;
        PL : REAL;
        VL : REAL;
        PVP : REAL;
        DPV : CData_REAL;
        AOFS : DWORD;
    END_STRUCT;
END_TYPE
```

This data structure defines all of the parameters within the block.
Some of these data items are actually data structures as well. For example, the PV is a data structure called `CData_REAL` and contains a series of values as follows:

```
TYPE
    CData_REAL : STRUCT
        Value : REAL;
        Status : DWORD;
        CInfo : DWORD;
        Dummy : DWORD;
        SH : REAL;
        SL : REAL;
        Unit : IndUnit;
    END_STRUCT;
END_TYPE
```

Therefore, when referencing the PV in the PVI block, the format is:

```
Identifier.PV.Value
```

Note that Unit is also a declared data type, although in this case it is an ARRAY, not a STRUCTURE:

```
TYPE
    IndUnit : ARRAY[0..7] OF BYTE;
END_TYPE
```
How to create a new data type:

Right-click on Data Types in the Project folder and select **Insert → Datatypes**:

2. Type in the name of the Data Type:
Double click on the Data Type and a blank page appears in the client area. The Edit Wizard has the three Data Types to choose from. The example below shows the result of selecting STRUCT (Data Structure):

```
1 TYPE
2  (*Typename*) :
3  STRUCT
4      (*Element 1 Name*) : (*DATATYPE*);
5      (*Element 2 Name*) : (*DATATYPE*);
6      (*Element 3 Name*) : (*DATATYPE*);
7      (*...*;
8      (*...*;
9      (*Element n Name*) ;
10     END_STRUCT;
11     END_TYPE
```

The green text is replaced with the required data names and types. The best way to work out what goes in here is to look at other examples (see previous pages).
1.5 Using Help in Logic Designer

The HELP function provides full online documentation and maybe accessed as a general document or as help on specific items.

**General Help documentation:**

- **Contents** – documentation about IEC61131 programming. Contains information about standard functions, function blocks, data types and variables, and configurations.

- **Help on FB/FU** – detailed documentation about each IEC61131 function and function block. Also covers function specific to the PROCON operating system for the Stardom controller.

- **PLC Help** – not implemented


- **PAS POU Help** – detailed documentation about the functions and function blocks provided by Yokogawa to operate in the Stardom controller.

**Context sensitive help**

Specific information about various items can be accessed by pressing F1 when the item is selected. Generally, this calls up the IEC61131 help document.

For detailed information about a function or function/block:

1. Right-click on the Function/Function Block in EDIT WIZARD
2. Select FB/FU
SECTION 2

IEC61131-3 PROGRAMMING

STRUCTURED TEXT
CONTENTS

2.1 Introduction ..............................................................................................................................3
2.2 Concepts ...................................................................................................................................4
  2.2.1 Assignment statements .......................................................................................................4
  2.2.2 Expressions ........................................................................................................................5
  2.2.3 Operators ............................................................................................................................6
  2.2.4 Expression evaluation ........................................................................................................7
2.3 Statements ................................................................................................................................8
  2.3.1 Calling Function Blocks ....................................................................................................8
  2.3.2 Conditional statements .....................................................................................................10
  2.3.3 Iteration statements ..........................................................................................................14
2.4 Structured Text Example ........................................................................................................17
2.1 Introduction

Structured Text is a high level language which has a comprehensive range of constructs for:

- Assigning values to variables,
- Calling functions and FBs
- Creating expressions
- Conditional evaluation of selected statements and for iteration

Structured Text has the following features:

- Developed specifically for industrial control applications
- Statements written in a fairly free style where tabs, line feed characters, comments can be inserted between keywords and identifiers
- Easy to read and understand, especially when written with meaningful identifiers and annotated with comments
2.2 Concepts

2.2.1 Assignment statements

Assignment Statements are used to change the value stored within a variable or the value returned by a function.

Assignment Statements have the following general format:

\[ X := Y; \] (* X and Y, the same data type*)

where \( Y \) represents an expression that produces a new value for the variable \( X \), when the statement is evaluated.

- The expression can be very simple, e.g. a literal constant
- or complex, e.g. involving nested expressions.
- The assigned variable can be a simple element or a multi-element variable such as an array or structure.

- Examples of assignment statements

  Rate := 13.1; (* literal value, constant *)
  Count := Count + 1; (* simple expression *)
  A1 := LOG(Rate); (* value from a function *)
  Alarm1.Time := RTC.CDT; (* value from a function block output parameter *)
  Profile[3] := 10.3 + SQRT((Rate + 2.0) * (A1/2.3)); (* value from a complex expression assigned to a single element of an array *)
2.2.2 Expressions

Expressions are used to calculate or evaluate values derived from other variables or constants and can involve one or more operators, variables and functions.

Expressions always produce a value of a particular data type which can be an elementary or derived data type.

Composite expressions can be built up by nesting simpler expressions. This allows ST to be used to perform complex arithmetic computations and data manipulation using different types of data.
### 2.2.3 Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>(..)</td>
<td>Parenthesized expression</td>
<td>Highest</td>
</tr>
<tr>
<td>Function(..)</td>
<td>Parameter list of a function, function evaluation</td>
<td></td>
</tr>
<tr>
<td>**</td>
<td>Exponentiation, i.e. raising to a power</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>Negation</td>
<td></td>
</tr>
<tr>
<td>NOT</td>
<td>Boolean complement, i.e. value with opposite sign</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td></td>
</tr>
<tr>
<td>MOD</td>
<td>Modulus operation</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>Addition</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td></td>
</tr>
<tr>
<td>&lt;, &gt;, &lt;=, =&gt;</td>
<td>Comparison operators</td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>Equality</td>
<td></td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Inequality</td>
<td></td>
</tr>
<tr>
<td>AND, &amp;</td>
<td>Boolean AND</td>
<td></td>
</tr>
<tr>
<td>XOR</td>
<td>Boolean exclusive OR</td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td>Boolean OR</td>
<td>Lowest</td>
</tr>
</tbody>
</table>

**Examples**

Volts := Amps * ohms;
Start := Oilpress AND Steam AND Pump;
2.2.4 Expression evaluation

Expressions are evaluated depending on the precedence of the operators and other sub-expressions, as follows:

- Bracketed expressions have the highest priority, followed by functions
- Highest precedence operators are evaluated first, followed by lower precedence, down to the lowest
- Where operators have the same precedence they are evaluated left to right

Consider the following example:

\[
\text{Rate} : = \text{Speed1}/10.0 + \text{Speed2}/(20.0-\sqrt{\text{Press}+6.0});
\]

(Assume Speed1 = 50.0, Speed2 = 60.0, Press = 30.0)

Evaluation order
- Rate : = 5.0 + 60/(20.0-6.0)
- Rate : = 5.0 + 60/14.0
- Rate : = 9.287

With Boolean operators the expression is only evaluated up to the point where the value can be resolved. For example:

\[
\text{StartUp} : = \text{A AND B AND D AND E};
\]

(where A, B, D and E are Boolean variables)

If A is false, then StartUp will be assigned the value FALSE, the rest of the expression will not be evaluated
2.3 Statements

Structured Text has a range of statements for:

- Calling Function Blocks (FBs)
- Conditional evaluation of statements
- Iteration

2.3.1 Calling Function Blocks

A Function Block instance can be invoked by:

- Calling the FB instance name with suitable input parameter values
- If parameter values are not provided
  - the current input parameters values as remaining from the last invocation are used
  - if the FB instance has not been invoked before, input parameters not assigned will take default values

Function Block instance invocation has the following general form:

FunctionBlockInstance(  
  InputParameter1 :=ValueExpression1,  
  InputParameter2 :=ValueExpression2,…);  

The ValueExpression should create a value of a data type that matches the FB input parameter.
**Example:**

```
VAR
    Loop1 : SimpleControl;
END_VAR

(* Initial invocation *)
    Loop1(PV := Input1.Out + Offset, SP := 100.0);
(* Further code........*).
(* Subsequent invocation*)
    Loop1(PV := Input1.Out + Offset, Mode := MANUAL);
```

The values of FB instance outputs can be:

- assigned to other variables
- assigned to a variable using ‘=> operator’

**Example:**

```
FlowRate := Loop1.Out;

VAR
    X12 : REAL;
END_VAR

Loop1(PV := %IW100, SP := 210.0, OUT => X12);
```
2.3.2 Conditional statements

Conditional Statements allow selected statements to be executed when certain conditions exist.

**IF...THEN...ELSE**

Selected ST statements can be evaluated depending on the value returned by a Boolean expression using the IF...THEN construct:

Format 1:

```
IF <Boolean expression> THEN
  <statements……>
END_IF;
```

Format 2:

```
IF<boolean expression> THEN
  < statements…>
ELSE
  < statements…>
END_IF;
```

The boolean expression can be any expression that returns a TRUE or FALSE boolean result.

**Example:**

```
IF Collision THEN
  Speed := 0;
  Brakes := ON;
END_IF;
IF (Gate = CLOSED) AND (Pump = ON) THEN
  Control_State := Active;
ELSE
  Control_State := Hold;
END_IF;
```
IF..THEN..ELSE constructs are also statements, and can be nested within other conditional statements to create complex conditional statements.

**Example:**

```
IF FlowRate > 230.0 THEN
  IF FlameSize > 4.0 THEN
    Fuel := 4000.0;
  ELSE
    Fuel := 2000.0;
  END_IF;
ELSE
  Fuel := 1000.0;
END_IF;
```

Further statements can be conditionally executed within the IF..THEN construct using the ELSEIF..THEN..ELSE construct which has the general form:

```
IF < boolean expression > THEN
  < statements…>.
ELSEIF < boolean expression > THEN
  < statements…..>
ELSE
  < statements…>
END_IF;
```

Any number of additional ELSEIF sections can be added to the IF…THEN construct

**Example:**

```
IF A>B THEN
  D := 1;
ELSEIF A = B+2 THEN
  D := 2;
ELSEIF A = B+3 THEN
  D := 4;
ELSE
  D := 3;
END_IF;
```
**CASE Statement**

The **Case** conditional statement is provided:

- So that selected statements can be executed depending on the value of an expression that returns an integer result.
- The integer expression should return a value of data type INT.
- The set of statements which have an integer selector value that matches the value of the integer expression are selected.

If no match is found, the statements preceded by ELSE are executed.

The CASE construct has the general form:

```
CASE < integer expression > OF
   < integer selector value1> : < statements….>  
   < integer selector value2 >: < statement……>
   ELSE
   < statements…>
END_CASE ;
```

**Example:**

```
CASE speed_setting OF
   1 : speed := 10.0;
   2 : speed := 20.4;
   3 : speed := 30.0;  fan1 := ON;
   4 : speed :=50.0;  fan2 := ON;
ELSE
   speed := 0;  SpeedFault := TRUE;
END_CASE;
```
Enumerated variables can be used in CASE statements:

**Example:**

```plaintext
TYPE
  SPEED : (STOPPED, SLOW, MEDIUM, FAST);
END_TYPE

VAR
  PumpSpeed : SPEED;
  Rate : REAL;
  Alarm AT %Q141 : BOOL;
END_VAR

CASE PumpSpeed OF:
  STOPPED : Rate := 0.0;
  SLOW : Rate := 20.4;
  MEDIUM : Rate := 30.4;
ELSE
  Rate := 0; Alarm := TRUE;
END_CASE;
```
2.3.3 Iteration statements

Used in situations where it is necessary to repeat one or more statements a number of times depending on the state of a particular variable or condition.

Be careful about the following issues when using iteration statements:

- Care should be taken to avoid situations that create endless loops
- They may also significantly increase the time to execute software elements e.g. FBs

**FOR..DO**

The FOR..DO construct allows a set of statements to be repeated depending on the value of an iteration variable:

- this is an integer variable of data type SINT, INT or DINT which is used to count statement executions
- can be used to count up or down using any sized increment until a final value is reached

This construct takes the general form:

```
FOR < initialize iteration variable >
TO < final value expression >
BY < increment expression > DO
  < statements….>
END_FOR;
```

The BY keyword is optional; if omitted, the iteration variable will increase by 1

**Examples:**

```
FOR I:= 100 TO 1 BY -1 DO
  channel [I]. Status := ON;
END_FOR;

FOR T: = FarmSize-1 TO TankMax*2 DO
  TankNo := T; vessel ( Tank := T);
END_FOR;
```
**WHILE..DO**

The WHILE...DO construct allows one or more statements to be executed while a particular boolean expression remains true.

- the boolean expression is tested prior to executing the statement
- if it is false the statements within the WHILE..DO are not executed

This construct takes the general form:

```
WHILE < boolean expression > DO
  < statements...>
END_WHILE
```

**Example:**

```
WHILE Value < (MaxValue - 10.0) DO
  bridge1();
  Value := Value + bridge1.Position;
END_WHILE
```

**REPEAT..UNTIL**

The REPEAT..UNTIL construct allows one or more statements to be executed while a particular boolean expression remains true.

- The boolean expression is tested after executing the statement
- if it is true, the statements within the REPEAT..DO are executed again

This construct takes the general form:

```
REPEAT
  < statements...>
UNTIL < boolean expression>
END_REPEAT
```

**Example:**

```
tries := 0;
REPEAT
  tries := tries + 1;
  switchgear1 ( Mode := DISABLE);
UNTIL (switchgear1.state = OFF) OR (tries > 4)
END_REPEAT
```
EXIT

This statement can only be used within iteration statements and allows iteration loops to be ended prematurely. When an EXIT statement is reached execution continues immediately from the end of the iteration construct. No further part of the iteration construct is executed.

```
Fault := FALSE;
FOR I:=1 TO 20 DO
  FOR J:=0 TO 9 DO
    IF FaultList [I,J] THEN
      FaultNo := I*10 + J;
      Fault := TRUE; EXIT;
      END_IF;
    END_FOR;
  END_IF;
END_FOR;
IF Fault THEN EXIT;
END_IF;
```

RETURN

The RETURN statement is used to return prematurely from the code. This statement causes execution to continue from the end of the function or FB body.

Note: The RETURN statement can only be used in function and FB bodies.

Example

```
FUNCTION_BLOCK TEST_POWER
VAR_INPUT
  CURRENT, VOLTS1, VOLTS2, VOLTS3 : REAL;
END_VAR
VAR_OUTPUT
  OVERVOLTS : BOOL;
END_VAR

IF VOLTS1*CURRENT > 100 THEN
  OVERVOLTS := TRUE; RETURN;
END_IF;
IF VOLTS2*(CURRENT + 10.0) > 100 THEN
  OVERVOLTS := TRUE; RETURN;
END_IF;
IF VOLTS3*(CURRENT + 20.0) > 100 THEN
  OVERVOLTS := TRUE; RETURN;
END_IF;
END_FUNCTION_BLOCK;
```
2.4 Structured Text Example

This example creates a function block using Structured Text to control a tank level. The function fills the tank, stirs the tank, and then empties the tank.

This example demonstrates some important issues regarding data conversion, which is critical to the success of developing the program.

It is assumed that the FCJ digital I/O connected to switches and LEDs.
Step 1 – Project Configuration:

1. Create a new project. Make sure to select Stardom PAS in the project creation wizard.
2. Save it as IECEX01.
3. Double-click on TargetSetting and set the IP address of your FCX.
4. Create 3 ST Function Blocks called:
   - TANK
   - INPUTSIM
   - OUTPUTSIM

Step 2 – I/O Configuration:

Double-click on DeviceLabelDefinition and configure the I/O as follows:

DI1  I_Sts  Task1
DI2  I_Sts  Task1
DI3  I_Sts  Task1
DI4  I_Sts  Task1
DO1  O_Sts  Task1
DO2  O_Sts  Task1
DO3  O_Sts  Task1
DO4  O_Sts  Task1

<table>
<thead>
<tr>
<th>Device label name</th>
<th>Comment</th>
<th>I/O Category</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>DI1</td>
<td>COMMAND</td>
<td>_Sts</td>
<td>Task1</td>
</tr>
<tr>
<td>DI2</td>
<td>COMMAND</td>
<td>_Sts</td>
<td>Task1</td>
</tr>
<tr>
<td>DI3</td>
<td>WEIGHT</td>
<td>_Sts</td>
<td>Task1</td>
</tr>
<tr>
<td>DI4</td>
<td>WEIGHT</td>
<td>_Sts</td>
<td>Task1</td>
</tr>
</tbody>
</table>
Step 3 – Programming the TANK Function Block:

First it is necessary to setup the variables for the block:

1. Double-click on TANKV

2. Insert the following variables:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Istate</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>State</td>
<td>STRING</td>
<td>VAR</td>
</tr>
<tr>
<td>Empty</td>
<td>BOOL</td>
<td>VAR</td>
</tr>
<tr>
<td>Fill</td>
<td>BOOL</td>
<td>VAR</td>
</tr>
<tr>
<td>Command</td>
<td>INT</td>
<td>VAR_INPUT</td>
</tr>
<tr>
<td>WeightA</td>
<td>REAL</td>
<td>VAR_INPUT</td>
</tr>
<tr>
<td>FullWeight</td>
<td>REAL</td>
<td>VAR_INPUT</td>
</tr>
<tr>
<td>EmptyWeight</td>
<td>REAL</td>
<td>VAR_INPUT</td>
</tr>
<tr>
<td>FillValve</td>
<td>DTag_O_Sts</td>
<td>VAR_OUTPUT</td>
</tr>
<tr>
<td>EmptyValve</td>
<td>DTag_O_Sts</td>
<td>VAR_OUTPUT</td>
</tr>
<tr>
<td>StirSpeed</td>
<td>REAL</td>
<td>VAR_OUTPUT</td>
</tr>
</tbody>
</table>

Variables with VAR_INPUT attribute will appear as input terminals on the Function Block. Variables with VAR_OUTPUT attribute will appear as output terminals on the Function Block.

Variables of with VAR attribute are local variables used within the POU only.
Double-click on TANK and type in the program as follows:

```plaintext
(* Check the vessel state *)
If WeightA >= FullWeight THEN  (* Is it full? *)
    State := 'FULL';
    Istate := 1;
ELSIF WeightA <= EmptyWeight THEN  (* or empty? *)
    State := 'EMPTIED';
    Istate := 2;
ELSE
    State := 'NOT_FULL';
    Istate := 3;
END_IF;

(* Process the command mode *)
CASE Command OF
1 : Empty := FALSE;  (* FILL TANK *)
    Fill := SEL(Istate = 1,TRUE,FALSE);
2 : Empty := FALSE;  (* HOLD contents *)
    Fill := FALSE;
4 : Fill := FALSE;  (* EMPTY Tank *)
    Empty := TRUE;
END_CASE;

(* Control the stirrer speed *)
StirSpeed := SEL((Command = 3)&(Istate = 1),100.0,0.0);

(* Convert Empty/Fill commands to Outputs *)
EmptyValve.Value := BOOL_TO_WORD (Empty);
FillValve.Value  := BOOL_TO_WORD (Fill);
```

Notes about the program:

1. The word “Weight” is a keyword and therefore cannot be used as a Variable. WeightA is used instead.
2. The first part of the program uses the IF THEN/ELSIF/ELSE construct.
3. The second part of the program uses the CASE OF construct.
4. The SEL command is used in the second and third parts of the program. This is a FUNCTION which returns one of two values based on the result of a boolean.
   - Example: FILL := SEL(Istate = 1,TRUE,FALSE);
     - FILL is a BOOL that will be assigned the values TRUE or FALSE depending on the result of “Istate = 1”. If the result of this is TRUE, then the first value (in this case “TRUE”) will be returned. Otherwise, the second value (“FALSE”) will be returned.
5. The digital outputs data types are “DTag_O_Sts”. This is a Data Structure, and the contents of this structure can be found in the following library:

SD_FIELD_PF - Data Types - DEVICE_TAG_TYPE

The value within this structure that sets the output is “value” of data type “WORD”.

Therefore, the BOOLEAN values, FILL and EMPTY, need to be converted to WORD variables using the BOOL_TO_WORD Function.

**Step 4 – The Input Simulation Program**

To provide some interface and simulation of the inputs, prepare a Structured Text Function Block called INPUTSIM.

1. Double-click on INPUTSIMV and setup the variables as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D11</td>
<td>DTag_O_Sts</td>
<td>VAR_INPUT</td>
<td></td>
</tr>
<tr>
<td>D12</td>
<td>DTag_O_Sts</td>
<td>VAR_INPUT</td>
<td></td>
</tr>
<tr>
<td>D13</td>
<td>DTag_O_Sts</td>
<td>VAR_INPUT</td>
<td></td>
</tr>
<tr>
<td>D14</td>
<td>DTag_O_Sts</td>
<td>VAR_INPUT</td>
<td></td>
</tr>
<tr>
<td>BD1</td>
<td>BOOL</td>
<td>VAR</td>
<td></td>
</tr>
<tr>
<td>BD2</td>
<td>BOOL</td>
<td>VAR</td>
<td></td>
</tr>
<tr>
<td>BD3</td>
<td>BOOL</td>
<td>VAR</td>
<td></td>
</tr>
<tr>
<td>BD4</td>
<td>BOOL</td>
<td>VAR</td>
<td></td>
</tr>
<tr>
<td>SWEIGHT</td>
<td>REAL</td>
<td>VAR_OUTPUT</td>
<td></td>
</tr>
<tr>
<td>SCOM</td>
<td>INT</td>
<td>VAR_OUTPUT</td>
<td></td>
</tr>
</tbody>
</table>

Four inputs are defined, and these interface directly to the Digital Inputs configured in the I/O configuration. Two outputs are defined that match the inputs of the TANK Function Block. The local variables are interfaces to the digital inputs (after conversion to BOOL).
2. Double-click on INPUTSIM and type in the following program:

```plaintext
1 ({* Convert Digital Inputs from Words to Booleans *})
2
3 BD1:=WORD_TO_BOOL(DI1.Value);  (* BOTH VALUES SHUT *)
4 BD2:=WORD_TO_BOOL(DI2.Value);
5 BD3:=WORD_TO_BOOL(DI3.Value);
6 BD4:=WORD_TO_BOOL(DI4.Value);
7
8 ({* Set Command based on DI1 and DI2 Digital Inputs *})
9
10 IF BD1 = FALSE & BD2 = FALSE then
11   SCOM := 2;  (* BOTH VALVES SHUT *)
12 ELSEIF BD1 = TRUE & BD2 = FALSE then
13   SCOM := 1;  (* FILL COMMAND *)
14 ELSEIF BD1 = FALSE & BD2 = TRUE then
15   SCOM := 4;  (* EMPTY COMMAND *)
16 ELSEIF BD1 = TRUE & BD2 = TRUE then
17   SCOM := 3;  (* STIR COMMAND *)
18 END_IF;
19
20 ({* Set Weight based on DI3 and DI4 Digital Inputs *})
21
22 IF BD3 = FALSE & BD4 = FALSE then
23   SWEIGHT := 0.0 ;
24 ELSEIF BD3 = TRUE & BD4 = FALSE then
25   SWEIGHT := 50.0 ;
26 ELSEIF BD3 = TRUE & BD4 = TRUE then
27   SWEIGHT := 100.0 ;
28 END_IF;
29
```

This program performs the following functions:

1. Convert the digital inputs from WORDs to BOOLs. These are easier to use in the program.

2. Calculates an Integer number for the output SCOM based on digital inputs 1 and 2, using the IF THEN construct.

3. Calculates a Real number for the output SWEIGHT based on the digital inputs 3 and 4, using the IF THEN construct.
Step 5 - The Output Simulation Program

To provide some interface and simulation of the outputs, prepare a Structured Text Function Block called OUTPUTSIM.

1. Double-click on OUTPUTSIMV and setup the variables as follows:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEED</td>
<td>REAL</td>
<td>VAR_INPUT</td>
</tr>
<tr>
<td>PTIME</td>
<td>TIME</td>
<td>VAR</td>
</tr>
<tr>
<td>OSPPEED</td>
<td>DINT</td>
<td>VAR</td>
</tr>
<tr>
<td>TP1START</td>
<td>BOOL</td>
<td>VAR</td>
</tr>
<tr>
<td>TP2START</td>
<td>BOOL</td>
<td>VAR</td>
</tr>
<tr>
<td>TP1EXP</td>
<td>BOOL</td>
<td>VAR</td>
</tr>
<tr>
<td>TP2EXP</td>
<td>BOOL</td>
<td>VAR</td>
</tr>
<tr>
<td>DO1</td>
<td>DTtag_O_Sts</td>
<td>VAR_OUTPUT</td>
</tr>
<tr>
<td>DO2</td>
<td>DTtag_O_Sts</td>
<td>VAR_OUTPUT</td>
</tr>
<tr>
<td>TP_1</td>
<td>TP</td>
<td>VAR</td>
</tr>
<tr>
<td>TP_2</td>
<td>TP</td>
<td>VAR</td>
</tr>
</tbody>
</table>

Two outputs are defined, and these interface directly to the Digital Outputs configured in the I/O configuration. One input is defined that matches the StirSpeed output of the TANK Function Block.
2. Double-click on OUTPUTSIM and type in the following program:

```
1. (* Read the analog Stir Speed value from the tank controller *)
2. (* and pulse two digital outputs *)
3. (* Set Pulse Time = 100/SPEED *)
4. (* First, the Speed input must be converted from REAL to TIME *)
5. (* This is a two step process because there is no direct conversion *)
6. (* from REAL to TIME *)
7. IF SPEED > 1.0 THEN (* This is to avoid dividing by 0 *)
8.   DSPEED := REAL_TO_DINT(100.0/SPEED);
9.   PTIME := DINT_TO_TIME(DSPEED);
10. END_IF;
11. PTIME := T#160s;
12. END_IF;
13. (* Pulse Outputs *)
14. (* Toggle Time 1 Start, i.e. When TP_Z expires, turn on TP_1 *)
15. TP1START := NOT TP1EXP ;
16. (* Pulse Output 1 according to TP_1 *)
17. IF_1(IN := TP1START, PT := PTIME);
18. IF1EXP := IF_1.Q;
19. (* Toggle Time 2 Start *)
20. TP2START := NOT TP2EXP ;
21. (* Convert Boolean to Digital outputs *)
22. DC1.Value := BOOL_TO_WORD(TP1START),
23. DC2.Value := BOOL_TO_WORD(TP2START);
```
This program is based on the Timer Function Block (TP), using two timers to cause two outputs to flash alternately and performs the following functions:

1. Calculate a pulse rate: \( 100.0 / \text{SPEED} \), that is, if \( \text{SPEED} \) is 100.0, then the pulse rate will be 1 second, and if the \( \text{SPEED} \) is 1.0, then the pulse rate will be 100 seconds.
2. To avoid dividing by 0, if the \( \text{SPEED} \) is less than 1.0, then fix the pulse rate at 100 seconds.
3. This pulse rate must be converted from a REAL variable to a TIME variable. This is done as a two step process because there is no direct conversion from REAL to TIME. The two functions (from the Type Conv. FUs group) REAL_TO_DINT and DINT_TO_TIME are used.
4. Create a Timer function called TP_1. This is set on when TP_2 expires according to the statement:
   \[
   TP1START := \neg TP2EXP
   \]
   TP1START is the trigger to start the TP_1 timer. TP2EXP goes off when TP_2 expires. The \( \neg \) inverts the sense of TP2EXP, so that TP1START goes ON when TP2EXP goes off.
   
   The timer (TP_1) has two inputs:
   
   IN is the input trigger to start the timer and is assigned to TP1START
   PT is the timer setpoint and is assigned to PTIME.

   The timer function is: if the IN trigger is on, the turn on the Output (Q) for time PT. After time PT, turn off the output. This output is loaded into the digital output DO1.

5. Create a timer function called TP_2. This is configured in the same way as TP_1, but turns on when TP_1 turns off. The output is loaded into digital output DO2.
6. The outputs are converted to WORDs as these are the values in the DTag_O_Sts data structure.
Step 6 - Create a Program using these function blocks

1. Double-click on MAIN and place the function block on the worksheet. The function blocks are in the IECEX01 group in the EDIT WIZARD.

2. Make the project, by selecting BUILD → MAKE.
3. Set up the digital I/O with the same device labels in the FCX using the Resource Configurator.
4. Run the project in Debugging mode.
SECTION 3

IEC61131-3 PROGRAMMING

FUNCTION BLOCK DIAGRAM
CONTENTS

3.1 Introduction.............................................................................................................................. 3
3.2 Graphical conventions within FBD.......................................................................................... 3
3.3 FBD Signal flow ...................................................................................................................... 5
3.4 FBD Feedback........................................................................................................................ 6
3.5 Function execution control .................................................................................................... 7
3.6 Jumps and labels .................................................................................................................... 9
3.7 Wireless Connections............................................................................................................. 10
3.8 Network evaluation rules ....................................................................................................... 11
3.9 Function Block Diagram Example......................................................................................... 12
### 3.1 Introduction

**Function Block Diagram** is a graphical language based on viewing a system in terms of the flow of signals between processing elements. It is similar to signal flows depicted in electronic circuit diagrams.

FBDs are used to express the behavior of functions, FBs and programs as a set of interconnected graphical blocks, the behavior of steps, actions and transitions in SFCs, and the flow of signals between processing elements.

### 3.2 Graphical conventions within FBD

- a FB is depicted as a rectangular block with inputs entering from the left and outputs exiting on the right
- FB type name is always shown within the block
- whereas the name of the FB instance is always shown above the block
- the formal names of FB inputs and outputs are shown within the block

**Function Block Diagram graphic representation**
Example 1 - Tank control Function Block Diagram example

LOADED_WT

32_HS_1234

>AVE_WEIGHT

2.5

TankControl1

TankControl

FillValue

Weight

EmptyValue

FullWeight

StirSpeed

EMPTY

STIR

Weigher1

Weigher

Offset

Weight

ADD

DIV

AVE_WEIGHT

2.0

EMPTY_ADJ

Weigher2

Weigher

Offset

Weight
3.3 FBD Signal flow

Signals are considered to flow:

- from the outputs of functions or FBs
- to the inputs of other functions or FBs

The outputs of FBs are updated as a result of FB evaluations hence changes of signal states and values propagate from left to right across the FBD network.

*When using boolean signals:*

- negated inputs of functions and FBs can be shown using a small circle at the input point
- similarly, negated outputs can be shown by placing a small circle at the output

*Example 2 - Negating boolean signals*
3.4 FBD Feedback

Example 3 - FBD feedback

An explicit feedback path is shown where the output Level from FB instance Load1 is fed back to the input ProcessValue of the preceding block Loop1.

Implicit feedback paths can also be created by using connectors that link feedback paths.

A feedback path implies that a value within the feedback path is retained after the FBD network is evaluated and used as the starting value on the next network evaluation.

There are two ways the feedback path can be evaluated:

- If FB instance Loop1 is evaluated before Load1 then at the start of each network evaluation, input ProcessValue will be set to the value of Load1. Level as retained from the previous evaluation.
- If FB instance Load1 is evaluated before Loop1 then input FlowRate will be set to the value of Loop1. Output as retained from the previous evaluation.

As the same FBD network can be evaluated in any order by a PLC the behavior of networks with feedback paths may vary subtly between implementations.
3.5 Function execution control

*Implicit and Explicit execution order*

By default, execution order is implicit from the position of the function in the network

- In the previous example, the function ‘\( \geq \)’ is evaluated after FB Loop1

This can be made explicit by using the function enable input EN

- EN is a boolean variable, that allows a function to be selectively evaluated

When the EN input is:

- **TRUE**, the function will be evaluated
- **FALSE**, the function will not be evaluated and its output value will not be generated

The function output ENO is a boolean variable that indicates that the function has completed its evaluation successfully.

- ENO value changes from FALSE to TRUE on completion of evaluation
Example 4 - Explicit function execution

As long as boolean variables add_Acid and pH_High are false functions MOVE and ADD are not evaluated.

When add_Acid and pH_High are both true:

- the EN input of the function MOVE is TRUE
- the value 100.0 is moved to variable stirRate
- when the function MOVE is evaluated, the ENO is asserted, this enables the ADD function
3.6 Jumps and labels

Jumps allow transfer control from one part of an FBD network to another using a graphical ‘jump’. Sections of FBD networks can be identified using labels.

By assigning a boolean signal to a label identifier, it is possible to transfer control to another part of the network, i.e. associated with the label.

Boolean signals that cause a jump in network evaluation order should be followed by a double arrow and the label identifier.

As there is ambiguity as to what happens to the rest of the network after a jump is made, **IEC 1131-3 recommends jumps in FBD not be used**

*Example 5 - FBD jump example*
3.7 Wireless Connections

Where it is inconvenient to connect variables using the standard wiring, the connector function allows data to be connected between function blocks within the same worksheet.

1. Click on the output terminal
2. Click on the Connector button
3. Type in a name, and click OK.
4. Follow same procedure for destination terminal
3.8 Network evaluation rules

The following ensure that data transferred between functions and FBs are consistent:

- No element in a network shall be evaluated unless the states of all inputs have been evaluated
- The evaluation of a network element shall not be complete until the states of all of its outputs have been evaluated

The evaluation of a network is not complete until the outputs of all its elements have been evaluated.

When data are transferred from one FBD network to another:

- all the values coming from elements in the first network should be produced by the same network evaluation
- the second network should not start evaluation until all values from the first network are available
3.9 Function Block Diagram Example

This example uses standard IEC functions and function blocks to create a Function Block diagram to control an air damper. The function of the FBD is as follows:

1. The signals ReqOpen and ReqClose signal the requested position for the damper. These requests are held in an RS bistable function block (a latch).
2. The output from this block drives the DemandOpen digital output. The output is inverted through a NOT function to drive the DemandClose digital output.
3. Limit Switches on the damper are input as digital inputs. These are compared using AND functions with the demand outputs.
4. The Open limit switch (OpenLS) is inverted and compared with the DemandOpen output through an AND function. If the demand is to open, and the damper is not open, the TON delay timer starts. If the damper does not open, the timer expires and the Discrepancy output is set on.
5. The Close limit switch (CloseLS) is inverted and compared to the DemandClose output through an AND function. If the demand is to close and the damper is not closed, the TON delay timer starts. If the damper does not close, the timer expires and the Discrepancy output is set on.
6. The TON time is set by the MoveTimeOut input.
Step 1 – Project Configuration:

1. Create a new project. Make sure to select Stardom PAS in the project creation wizard.
2. Save it as IECEX02.
3. Double-click on TargetSetting and set the IP address of your FCX.
4. Create an FBD Function Block called DAMPER.

Step 2 – I/O Configuration:

Double-click on DeviceLabelDefinition and configure the I/O as follows:

<table>
<thead>
<tr>
<th>Device label name</th>
<th>Comment</th>
<th>I/O Category</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>DI1</td>
<td>I_Sts</td>
<td>Task1</td>
<td></td>
</tr>
<tr>
<td>DI2</td>
<td>I_Sts</td>
<td>Task1</td>
<td></td>
</tr>
<tr>
<td>DI3</td>
<td>I_Sts</td>
<td>Task1</td>
<td></td>
</tr>
<tr>
<td>DI4</td>
<td>I_Sts</td>
<td>Task1</td>
<td></td>
</tr>
<tr>
<td>DO1</td>
<td>O_Sts</td>
<td>Task1</td>
<td></td>
</tr>
<tr>
<td>DO2</td>
<td>O_Sts</td>
<td>Task1</td>
<td></td>
</tr>
<tr>
<td>DO3</td>
<td>O_Sts</td>
<td>Task1</td>
<td></td>
</tr>
</tbody>
</table>
Step 3 – Programming the DAMPER Function Block

Double click on DAMPER, and place the functions and function blocks as follows:

Step 4 – Create a Program using this Function Block

Double-click on MAIN and place this Function Block on the worksheet.

Connect the digital I/O to the inputs and outputs of the block. Use the **BOOL** variable of each Digital I/O point to match the variable types of the function block.

Connect a time value (say, T#10s) to the MoveTimeOut input.
SECTION 4

IEC61131-3 PROGRAMMING

LADDER DIAGRAMS
## CONTENTS

4.1 CONCEPTS ................................................................................................................................. 3

4.2 METHODOLOGY ......................................................................................................................... 5

4.2.1 CIRCUIT OPERATION............................................................................................................... 5
4.2.2 CONNECTING FUNCTION BLOCKS IN LADDER DIAGRAMS ...................................................... 6
4.2.3 CONNECTING FUNCTIONS IN LADDER DIAGRAMS ............................................................... 7
4.2.4 FEEDBACK PATHS .................................................................................................................. 8
4.2.5 LD JUMPS AND LABELS ........................................................................................................ 8
4.2.6 LD NETWORK EVALUATION RULES ..................................................................................... 9

4.3 PROCEDURES .......................................................................................................................... 10

- Procedure 1 – Creating ladder diagrams ....................................................................................... 11
- Procedure 2 – Shortcut for creating a ladder rung ........................................................................... 12
- Procedure 3 – Adding contacts and coils to the rung ....................................................................... 12
- Procedure 4 – Creating parallel contact/coil connections ................................................................. 13
- Procedure 5 – Setting properties of the contacts and coils ............................................................... 14

4.4 LADDER DIAGRAM EXAMPLE ............................................................................................... 15
4.1 Concepts

Ladder Diagrams are a graphical language that uses ‘relay ladder’ logic symbols.

A Ladder Diagram has a *left hand vertical power rail* that notionally supplies power through contacts spread out along *horizontal rungs*. Each contact represents the state of a boolean variable.

When all contacts in a horizontal rung are made, i.e. in their ‘true’ state:

- Power can flow along the rail and operate a coil on the right of the rung
- Contacts which are normally open present a ‘true’ state when the contacts are closed
- Conversely contacts which are normally closed present a ‘true’ state when the contacts are opened

A **contact** provides a ‘read only’ access to the state of the variable. On the other hand a **coil** provides a ‘write only’ access and updates the state of the associated variable when notional power flow occurs.

The use of the right hand power rail is optional.

The network associated with the logic for one coil is called a ‘**ladder rung**’.
### Symbols used in Ladder Diagrams:

<table>
<thead>
<tr>
<th>Description</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Normally Open Contact</strong></td>
<td><img src="CO08" alt="Symbol" /></td>
</tr>
<tr>
<td>- power flow occurs from left to right when the associated variable is 1.</td>
<td></td>
</tr>
<tr>
<td><strong>Normally Closed Contact</strong></td>
<td><img src="CO10" alt="Symbol" /></td>
</tr>
<tr>
<td>- power flow occurs from left to right when the associated variable is 0.</td>
<td></td>
</tr>
<tr>
<td><strong>Coil</strong></td>
<td><img src="CO09" alt="Symbol" /></td>
</tr>
<tr>
<td>The coil is set to the state according to the power flow coming from the left hand link. If power flow is ON, the coil state is set ON.</td>
<td></td>
</tr>
<tr>
<td><strong>Negated Coil</strong></td>
<td><img src="CO11" alt="Symbol" /></td>
</tr>
<tr>
<td>The negated coil is set to the opposite state according to the power flow coming from the left hand link. If power flow is ON, the coil state is set OFF.</td>
<td></td>
</tr>
<tr>
<td><strong>SET Coil</strong></td>
<td><img src="CO13" alt="Symbol" /></td>
</tr>
<tr>
<td>The coil is reset to the ON state when there is power flow coming from the left hand link. The coil remains set until it is RESET.</td>
<td></td>
</tr>
<tr>
<td><strong>RESET Coil</strong></td>
<td><img src="CO15" alt="Symbol" /></td>
</tr>
<tr>
<td>The coil is set to the OFF state when there is power flow coming from the left hand link. The coil remains OFF until it is RESET.</td>
<td></td>
</tr>
</tbody>
</table>
4.2 Methodology

4.2.1 Circuit Operation

From the nature of electrical circuits:

- Contacts in a horizontal rung represent logical **AND** operations
- Alternative paths from the main rung provided by vertical paths are logical **OR** operations

To construct an **AND** in Ladder Logic, the contacts (conditions) are connected in series. All contacts must be made (i.e. all conditions must be met) for the ‘power’ to flow through the circuit and set the coil on.

To construct an **OR** in Ladder Logic, the contacts are connected in parallel. If any of the contacts are made, then power flows through to the coil.

**Example:**

```
Structured Text Equivalent:
X1: = (A1 OR B1 AND A2 AND A3) OR (C1 AND C2);
```

**Function Block Equivalent:**

![Diagram of Ladder Logic circuit with contacts and coils]
4.2.2 Connecting Function Blocks in Ladder Diagrams

FBs can be connected into Ladder Diagram rungs provided:

- They have boolean inputs and outputs
- Inputs can be directly driven from LD rungs
- Outputs can provide power flows for driving coils
4.2.3 Connecting Functions in Ladder Diagrams

- may be provided with an additional boolean input EN and output ENO for use in LD

- EN provides a power flow signal into the function
  - When EN is ON i.e. 1, the function is enabled and can evaluate its inputs to produce an output
  - When the evaluation is complete, ENO can be used to provide power flow to other functions or outputs

*Example:*

1. The EN of the AND function is driven by the left power rail, and so that AND function is always executing.

2. The inputs into the AND function do not have to be boolean, and in this case are DWORDs (note that the input type to an AND function is ANY_BIT).

3. The enable is propagated to the EN of the NE function, so that it is also always executing.

4. The inputs to the NE are ANY_NUM, and in this case are DWORDs. However, the output of a NE is a BOOL, and so it can connect into the ladder rungs. In this case it drives the coil ‘AUTSTAT’.
4.2.4 Feedback paths

It is possible to create Ladder diagram rungs which contain feedback loops where one or more values used in contacts, and as function and FB inputs, come from variables that are updated when the rungs are evaluated.

Example:

![Diagram showing feedback paths]

4.2.5 LD Jumps and Labels

Using a graphical jump it is possible to transfer control from one part of an LD network to another identified by a label.

Example

![Diagram showing LD jumps and labels]
4.2.6 LD network evaluation rules

PLC ladder rungs are always scanned from the top of a ladder diagram to the bottom.

The following rules ensure that the results are unambiguous and consistent:

- No element in a network shall be evaluated unless the states of all inputs have been evaluated
- The evaluation of a network element shall not be complete until the states of all its outputs have been evaluated
- The evaluation of a network is not complete until the outputs of all its elements have been evaluated
- When data are transferred from one network to another:
  - all the values coming from elements in the first network should be produced by the same network evaluation
  - the second network should not start evaluation until all values from the first network are available
4.3 Procedures

Ladder Diagrams can be created in a Ladder Diagram (LD) Function Block, or placed directly on a Function Block Diagram (FBD). They can also be placed as transition logic on an SFC.

The Ladder editing toolbar contains the following commands:

From left to right, these commands are:

- Contact Network - Create a Ladder rung
- Insert a contact (to the left)
- Save the ladder diagram
- Insert a coil
- Insert a contact (to the right)
- Add a contact/coil (above)
- Add a contact/coil (below)
- Add a left power rail
- Add a right power rail
- Insert an LD branch
- Toggle properties of contact/coil
Procedure 1 – Creating ladder diagrams

Individual contacts and coils can be applied to the worksheet, along with left and right power rungs and wired together using the wiring tool:

1. Click somewhere on the worksheet and click on the ‘Add Contact Right’ button. A contact will appear. Repeat this as many times as necessary to create the required number of contacts.

2. To add a coil repeat the above process, clicking on the ‘Add Coil Right’ button.

3. Wire the contacts and coils together using the wiring tool.

4. Connect the left most contacts to the left power rail by clicking on the contact then clicking on the ‘Left powerrail’ button. Contacts with separate power rails can be joined up so that they are all on the same rail.

5. Connect the coil(s) to the right power rail by clicking on the coil then clicking on the ‘Right powerrail’ button. Coils with separate power rails can be joined up so that they are all on the same rail.
Procedure 2 – Shortcut for creating a ladder rung

1. Open the LD worksheet and click somewhere on the left side of the sheet.

2. Click on the ‘Contact Network’ button. A simple ladder rung will appear with left and right power rails, a contact and a coil.

To add a second rung, click on the left power rail and click on the ‘Contact Network’ button.

Procedure 3 – Adding contacts and coils to the rung

To insert a contact to the right of an existing contact or to the right of a selected rung section, select the contact and click on the ‘Add Contact Right’ button. The new contact will appear on the rung to the right of the existing contact or rung section.

To insert a contact to the left of an existing contact or rung section, select the contact and click on the ‘Add Contact Left’ button. The new contact will appear on the rung to the left of the existing contact or rung section.

Note that the ‘Add Contact Left’ button is grayed out until a contact or rung is selected.
Procedure 4 – Creating parallel contact/coil connections

Using Procedure 1 it is possible to create parallel contact connections by creating contacts and wiring them into the ladder rungs.

However, there are shortcuts to help speed up this process:

To add a contact/coil directly in parallel with one existing contact/coil, select the contact on the rung and click on the ‘Add contact/coil below’ or ‘Add contact/coil above’ button. Extra contacts can be added to either side of this connection by clicking on the ‘Add Contact left/right’ buttons.

To add a contact in parallel with a number of contacts:

1. click on the ‘Insert LD branch’ button.
2. Click on the ladder rung so that a connection is made.
3. Drag the mouse and click to place the new contact.
4. Move the mouse to the destination rung location and click to connect the branch to the rung.
Procedure 5 – Setting properties of the contacts and coils

The properties of a contact or coil can be set by right clicking and selecting **Object Properties**.

The type of contact or coil can be set by selecting the contact/coil and clicking on the toggle button. This toggles through the various contact or coil types as described in section 4.1 (page 4).
4.4 Ladder Diagram Example

This exercises makes use of the damper function block created in the FBD section. Ladder diagram Function Blocks are created to check the interlocks of the dampers, and duty standby logic to drive one damper or the other.

Step 1 – Create the POUs

Open the IECEX02 project and deleted the DAMPER function block from the MAIN worksheet.

Create the following Ladder Diagram POUs with Function Block attribute:

- DamperCtls
- DTYSTBY

Add extra I/O in the DeviceLabelDefinition file so that there are at least 8 digital inputs and 4 digital outputs.
Step 2 - The interlock function

Open the DamperCtls worksheet and create the Ladder Diagram as shown below.

If power to the damper is on for more than 5 seconds
and the actuator is not in local
and it is racked-in,
Then set the Interlock output ON

If the Start input is ON,
send set ON a latch. – Send a pulse to the latch (2 sec)
If the Stop input is ON,
then reset the latch - Send a pulse to the latch (2 sec)
If the latch is on, and the Interlock is ON,
then set on the START/STOP output.

Variables:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>POWER</td>
<td>BOOL</td>
<td>VAR_INPUT</td>
</tr>
<tr>
<td>LOCAL</td>
<td>BOOL</td>
<td>VAR_INPUT</td>
</tr>
<tr>
<td>RACKIN</td>
<td>BOOL</td>
<td>VAR_INPUT</td>
</tr>
<tr>
<td>START</td>
<td>BOOL</td>
<td>VAR_INPUT</td>
</tr>
<tr>
<td>STOP</td>
<td>BOOL</td>
<td>VAR_INPUT</td>
</tr>
<tr>
<td>V006</td>
<td>BOOL</td>
<td>VAR</td>
</tr>
<tr>
<td>DPINT</td>
<td>BOOL</td>
<td>VAR_OUTPUT</td>
</tr>
<tr>
<td>DPSS</td>
<td>BOOL</td>
<td>VAR_OUTPUT</td>
</tr>
</tbody>
</table>

These two rungs could be combined into 1 rung.

These are function blocks with boolean I/O.
Step 3 – The Duty/Standby Function

Open the DTYSTDBY worksheet and create the Logic Diagram as shown below.

If Interlock input (D1INT) from Damper1 is ON
then set ON the Damper1 Duty output (D1DTY)
If Interlock input (D2INT) from Damper2 is ON and Damper1 Duty is not ON
then set ON the Damper2 Duty output (D2DTY)

If either Damper1 Duty OR Damper 2 Duty is ON,
then set ON the Damper Available output (DAVAIL)

If Start input is ON, AND Damper 1 Duty is ON,
then set Damper 1 Start command ON
If Start input is ON, AND Damper 2 Duty is ON,
then set Damper 2 Start command ON

Variables:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1INT</td>
<td>BOOL</td>
<td>VAR_INPUT</td>
</tr>
<tr>
<td>D2INT</td>
<td>BOOL</td>
<td>VAR_INPUT</td>
</tr>
<tr>
<td>DSTART</td>
<td>BOOL</td>
<td>VAR_INPUT</td>
</tr>
<tr>
<td>D1DTY</td>
<td>BOOL</td>
<td>VAR_OUTPUT</td>
</tr>
<tr>
<td>D2DTY</td>
<td>BOOL</td>
<td>VAR_OUTPUT</td>
</tr>
<tr>
<td>DAVAIL</td>
<td>BOOL</td>
<td>VAR_OUTPUT</td>
</tr>
<tr>
<td>D1START</td>
<td>BOOL</td>
<td>VAR_OUTPUT</td>
</tr>
<tr>
<td>D2START</td>
<td>BOOL</td>
<td>VAR_OUTPUT</td>
</tr>
</tbody>
</table>
Step 4 – Creating the Program

Open the MAIN worksheet and place the following function blocks on the worksheet:

<table>
<thead>
<tr>
<th>Name</th>
<th>Function Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>DamperCtrls_1</td>
<td>DamperCtrls</td>
</tr>
<tr>
<td>DamperCtrls_2</td>
<td>DamperCtrls</td>
</tr>
<tr>
<td>DTYSTBY_1</td>
<td>DTSTBY</td>
</tr>
<tr>
<td>DAMPER_1</td>
<td>DAMPER</td>
</tr>
<tr>
<td>DAMPER_2</td>
<td>DAMPER</td>
</tr>
</tbody>
</table>

Connect the blocks as shown on the following diagram:

Note that not all of the inputs and outputs are connected. This can be done so if desired. Otherwise, their values can be simulated when running in debug mode.

To run the program:

1. MAKE the project (BUILD menu).
2. DOWNLOAD the project (ONLINE menu – Project Control)
3. Set it into debugging mode (ONLINE menu)

The I/O states can be viewed and forced for the Program and each Function Block.
# CONTENTS

5.1 CONCEPTS...................................................................................................................................................3

5.2 IL INSTRUCTION SEMANTICS..................................................................................................................4
   5.2.1 SYNTAX ..............................................................................................................................................4
   5.2.2 DEFERRED EXECUTION .....................................................................................................................5
   5.2.3 MODIFIERS.......................................................................................................................................6

5.3 OPERATORS................................................................................................................................................7
   5.3.1 MAIN IL OPERATORS.........................................................................................................................7
   5.3.2 IL COMPARISON AND JUMP OPERATORS.......................................................................................8
   5.3.3 IL AND OPERATORS WITH MODIFIERS..........................................................................................8

5.4 METHODOLOGY.......................................................................................................................................9
   5.4.1 CALLING FUNCTION BLOCKS ...........................................................................................................9
   5.4.2 CALLING FUNCTIONS.......................................................................................................................10

5.5 PORTABILITY BETWEEN IL AND OTHER LANGUAGES .................................................................11

5.6 INSTRUCTION LIST EXAMPLE...............................................................................................................12
5.1 Concepts

**Instruction List** is a low-level language that is similar to a simple machine assembler. It is used to express functions, FBs and programs and also used to describe the actions and transitions of SFCs.

The language structure consists of a series of instructions, where each instruction is on a new line.

An instruction consists of an **operator** followed by **one or more operands**

- an operand is the subject of the operator
- operands represent variables or constants
- a few operators can take a series of operands

Each instruction may either use or change the value stored in a single register

- the standard refers to this register as the **result** of an instruction
- the **result** can be overwritten with a new value, modified or stored in a new variable
- the **result** register is sometimes called an **accumulator** or ‘**accu**’

IL also provides comparison operators:

- which can compare the value of a variable with the register value, i.e. with the current result
- the result of the comparison is written back to the register

Various types of jump instruction are provided that can test the current register value and, if required, jump to a named label. Labels can be used to identify various entry points for jump instructions. Each label should be followed by a colon.

Comments can only be placed at the end of the line, not at the beginning or between an operator or operand.

---

### Equivalent ST statements

<table>
<thead>
<tr>
<th>IL Instruction</th>
<th>ST Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD Speed</td>
<td><code>load speed 6</code></td>
</tr>
<tr>
<td>GT 1000</td>
<td><code>test if &gt; 1000?</code></td>
</tr>
<tr>
<td>JMP CN Volts_OK</td>
<td><code>jump if not</code></td>
</tr>
<tr>
<td>LD Volts</td>
<td><code>load volts 6</code></td>
</tr>
<tr>
<td>SUB 10</td>
<td><code>reduce by 10</code></td>
</tr>
<tr>
<td>ST Volts</td>
<td><code>store volts</code></td>
</tr>
<tr>
<td>Volts_OK :=</td>
<td></td>
</tr>
<tr>
<td>LD 1</td>
<td><code>load 1 &amp; store</code></td>
</tr>
<tr>
<td>ST %Q75</td>
<td><code>in output 75</code></td>
</tr>
</tbody>
</table>

---

IF Speed > 1000 THEN
Volts := Volts - 10;
END_IF;
%Q75 := 1;
5.2 IL instruction semantics

5.2.1 Syntax

An IL instruction for the majority of operators is:

NewResult := CurrentResult Operator Operand

Format:
1 Operator Operand

‘CurrentResult’ is the value currently held in the ‘result’ register:

- prior to the instruction being executed
- it is the value left over from the previous instruction

‘NewResult’ is the new value loaded in the ‘result’ register:

- as produced by executing the instruction
- it is the value produced by the ‘Operator’ acting on the values of ‘CurrentResult’ and the ‘Operand’

The ‘NewResult’ becomes the ‘CurrentResult’ for the next instruction

Example

SUB 10

function:
NewResult := CurrentResult SUB 10

A few IL operators do not fit the general model:

- For example, the ST (store) operator can be described by the expression:
  Operand := CurrentResult

  format       ST Operand

- This is because the function of the operator ST is to store the current value held in the result register
5.2.2 Deferred Execution

Parenthesis allow sections of IL instructions to be deferred, i.e. produce intermediate results that don’t affect the ‘result register’.

**Example**

```
LD  A  (* add A to B*)
ADD B  (* hold the value in result reg.*)
MUL( A  (* defer MUL until (A-B) available*)
    SUB B
)   (* now multiply by (A-B)*)
```

The parenthesis operators provide a similar function to a stack. A number of deferred operators can be active at any time allowing complex nested operations

**Example**

```
LD  X  (* load X 1*)
ADD( B  (* defer ADD, load B 2*)
    MUL( C  (* defer MUL, load C 3*)
        ADD D (* add D 4*)
    )  (* multiply result 5*)
)  (* add result 6*)
```

Equivalent ST  \[ X+(B*(C+D)) \]

The Stack operation:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Top</th>
<th>Top-1</th>
<th>Top-2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>result reg.</td>
<td>result reg.</td>
<td>result reg.</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>X</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>X</td>
<td>B</td>
<td>C+D</td>
</tr>
<tr>
<td>5</td>
<td>X</td>
<td>B*(C+D)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>X+(B*(C+D))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5.2.3 Modifiers

Some IL operators can take single letter modifiers after the operator mnemonic, that change the semantics of the instruction.

If the operand contains a boolean value, the ‘N’ modifier can be used to negate its value.

With jump operators, the ’N’ modifier indicates that the jump will use the negated value of the ‘result register’.

The ‘C’ modifier can be used only with jump operators and indicates that a jump is made conditionally on the boolean value of the ‘result register’.

Example:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operand</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>%IX10</td>
<td>(* load input 10*)</td>
</tr>
<tr>
<td>ANDN</td>
<td>Switch1</td>
<td>(* AND NOT switch1*)</td>
</tr>
<tr>
<td>JMPNC</td>
<td>Lab1</td>
<td>(<em>Jump if Not true</em>)</td>
</tr>
</tbody>
</table>
5.3 Operators

5.3.1 Main IL Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Modifiers</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>N</td>
<td>ANY(*1)</td>
<td>Load operand into result register</td>
</tr>
<tr>
<td>ST</td>
<td>N</td>
<td>ANY</td>
<td>Store result register into operand</td>
</tr>
<tr>
<td>S</td>
<td>Note 2</td>
<td>BOOL</td>
<td>Set operand true</td>
</tr>
<tr>
<td>R</td>
<td>Note 2</td>
<td>BOOL</td>
<td>Reset operand false</td>
</tr>
<tr>
<td>AND</td>
<td>N,(</td>
<td>ANY</td>
<td>Boolean AND</td>
</tr>
<tr>
<td>&amp;</td>
<td>N,(</td>
<td>ANY</td>
<td>Boolean AND</td>
</tr>
<tr>
<td>OR</td>
<td>N,(</td>
<td>ANY</td>
<td>Boolean OR</td>
</tr>
<tr>
<td>XOR</td>
<td>N,(</td>
<td>ANY</td>
<td>Boolean exclusive OR</td>
</tr>
<tr>
<td>NOT</td>
<td></td>
<td>ANY</td>
<td>Logical negation (invert logical input)</td>
</tr>
<tr>
<td>ADD</td>
<td>(</td>
<td>ANY</td>
<td>Addition</td>
</tr>
<tr>
<td>SUB</td>
<td>(</td>
<td>ANY</td>
<td>Subtraction</td>
</tr>
<tr>
<td>MUL</td>
<td>(</td>
<td>ANY</td>
<td>Multiplication</td>
</tr>
<tr>
<td>DIV</td>
<td>(</td>
<td>ANY</td>
<td>Division</td>
</tr>
<tr>
<td>MOD</td>
<td>(</td>
<td>ANY</td>
<td>Modulo-division (returns remainder)</td>
</tr>
</tbody>
</table>

Note 1: Operators that can take operands of data type ANY are said to be ‘overloaded’.

Note 2: The Set and Reset operators can only be used with operands of boolean data type.
5.3.2 IL comparison and jump operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Modifiers</th>
<th>Operand</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT</td>
<td>(</td>
<td>ANY(*)</td>
<td>Comparison – Greater Than</td>
</tr>
<tr>
<td>GE</td>
<td>(</td>
<td>ANY</td>
<td>Comparison – Greater than or equal to</td>
</tr>
<tr>
<td>EQ</td>
<td>(</td>
<td>ANY</td>
<td>Comparison – Equal</td>
</tr>
<tr>
<td>NE</td>
<td>(</td>
<td>ANY</td>
<td>Comparison – Not Equal</td>
</tr>
<tr>
<td>LE</td>
<td>(</td>
<td>ANY</td>
<td>Comparison – Less than or equal to</td>
</tr>
<tr>
<td>LT</td>
<td>(</td>
<td>ANY</td>
<td>Comparison – Less than</td>
</tr>
<tr>
<td>JMP</td>
<td>C, N</td>
<td>LABEL</td>
<td>Jump to Label</td>
</tr>
<tr>
<td>CAL</td>
<td>C, N</td>
<td>NAME</td>
<td>Call Function Block</td>
</tr>
<tr>
<td>RET</td>
<td>C, N</td>
<td></td>
<td>Return from Function or Function Block (*4)</td>
</tr>
<tr>
<td>)</td>
<td></td>
<td></td>
<td>Execute the last deferred operator</td>
</tr>
</tbody>
</table>

Note 3: Operators that can have more than one modifier may be used with either or both modifiers. See below for an example using the AND operator.

Note 4: The RET operator is used to return from a function or function block. The modifier C means that the return is made conditionally.

5.3.3 IL AND operators with modifiers

<table>
<thead>
<tr>
<th>Operator</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>Boolean AND</td>
</tr>
<tr>
<td>AND(</td>
<td>Deferred boolean AND</td>
</tr>
<tr>
<td>ANDN</td>
<td>Invert boolean AND</td>
</tr>
<tr>
<td>ANDN(</td>
<td>Deferred boolean AND, invert deferred result</td>
</tr>
</tbody>
</table>
5.4 Methodology

5.4.1 Calling Function Blocks

The standard provides three formats of the CAL operator for calling FBs:

- calling a FB using a formal call with an input list
- calling a FB using an informal call
- calling FBs using input operators

Calling a FB using a formal call with an input list

- The CAL operator is followed by a list of FB parameters with their values.
- Each input parameter must be identified by name
- The value of each parameter can be given directly or calculated and passed into the IL accumulator.

Example:

```
CAL LOOP1 ( 
  SP := 300.0 
  PV := ( 
    LD %IW20 
    ADD 10 
  ) )
```

Calling a FB using an informal call

The values to each input should be set-up prior to calling the FB.

Example:

```
LD 300.0 
ST LOOP1.SP 
LD %IW20 
ADD 10 
ST LOOP1.PV 
CAL LOOP1
```

In both the above formats, the output(s) of the function block are stored in their variables, which can then be accessed using the LD function.
Calling FBs using input operators

- Can only be used with some IEC standard FBs.
- A number of IL operators are reserved for use with commonly used FBs such as SR (Set/Reset) bistable or the CTU, up-counter.

**Example:**

```
S1 Latch1 (* Latch1 an FB instance of SR*)
LD 10 (* S1 operator used only with SR*)
PV CTU1
CU CTU1
```

In this case, S1 causes Latch1 to be set, PV is loaded with 10 and CU calls the FB CTU1 to start counting.

### 5.4.2 Calling Functions

**Calling functions using a formal call**

In a formal function call, the function name is given, followed by the values of the input parameters.

**Example:**

```
SHR(IN := %IW30, N := 10)
```

**Calling functions using an informal call**

In an informal function call, the value of the first parameter is supplied by the current result. The function is given, followed by the values of the input parameters.

**Example:**

```
LD %IW30
SHR 10
ST %QW10
```

The SHR (shift right) function requires two input parameters:

- In the above example, the first parameter is loaded from the current result, i.e. the contents of %IW30
- The second parameter has value 10, the count of bits for the shift operation.
5.5 Portability between IL and other languages

The conversion of sections of Instruction List into the other languages is very difficult. It could be achieved if a restricted range of IL operators were used and the instructions written using a strict format.

*Example:*

This can be written in Instruction List as follows:

```
LD  Test1    (* Test1 or *)
OR  Test2    (* Test2 *)
AND Sw1      (* AND Sw1 *)
AND %IX3     (* AND input 3 *)
ST  StartSR.S1 (* Set input of StartSR *)
LD  Reset    (* Load value of Reset *)
ST  StartSR.R (* Store in reset input *)
CAL StartSR  (* Call fb. StartSR *)
LD  StartSR.Q1 (* Load output Q1 *)
ST  Start    (* and store in Start *)
```
5.6 Instruction List Example

This example is based on the example used for Structured Text (section 2). It is created in the same project as the ST example and used in MAIN to enable comparison of the two blocks.

Step 1 – Project setup

1. Open project IECEX01.

2. Create an IL Function Block called TANKIL.

3. Copy the variables from TANK to TANKIL:
   1. Double-click on TANKV.
   2. Select all of the variables and COPY.
   3. Double-click on TANKILV.
   4. Insert a variable, select it, and then PASTE.
   5. Delete the inserted variable.

4. Make sure that there are enough I/O variables defined in the DeviceLabelDefinition file for the extra I/O (i.e. a total of 8 digital input and 8 digital outputs).
Step 2 – Write the IL Program

```plaintext
LD WeightA
GE FullWeight (* Is WeightA >= FullWeight? *)
JMP CN EW (* If not, Jump to EW *)
LD 'FULL'
ST State (* If it is, set State = FULL *)
LD 1
ST Istate (* and Istate = 1 *)
JMP COMM (* Goto COMM *)

EW:
LD WeightA
LE EmptyWeight (* Is WeightA <= FullWeight? *)
JMP CN NW (* If not, Jump to NW *)
LD 'EMPTIED'
ST State (* If it is, set State = EMPTIED *)
LD 2
ST Istate (* and Istate = 2 *)
JMP COMM (* Goto COMM *)

NW:
LD 'NOT_FULL'
ST State (* Otherwise, set State = NOT_FULL *)
LD 3
ST Istate (* and Istate = 3 *)
COMM: (* Command execution *)
LD FALSE
ST Empty (* Initialize the Empty Command *)
ST Fill (* and the Fill Command *)
LD Command (* Is Command = 1? *)
EQ 1
JMP CN EMPTYC (*If not, jump to EMPTYC *)
LD Istate
EQ 1 (* Does Istate = 1?*)
SEL TRUE,FALSE
ST Fill (* Set Fill = TRUE/FALSE using SEL function *)
JMP STIR (* Jump to stirring section *)

EMPTYC:
LD Command (* Is Command = 4? *)
EQ 4
JMP CN STIR (* If not, jump to STIR *)
LD TRUE
ST Empty (* Set the Empty Command = TRUE *)
STIR:
LD Command
EQ 3 (*If Command = 3 *)
AND( Istate
EQ 1 (* AND Istate = 1 *)
)
SEL 100.0,0.0
ST StirSpeed (* Set StirSpeed = 100.0, Else set to 0.0 *)
```

Step 3 – Creating and Running the Program

1. Save and compile the Instruction List program.

2. Double-click on MAIN, and place the TANKIL Function Block on the worksheet.

3. Connect the Input and Output function blocks to the TANKIL block.

4. Connect digital I/O to the input/output function blocks.

5. MAKE (BUILD menu) and DOWNLOAD (ONLINE menu) the project.

6. Run the project in Debug mode (ONLINE menu).
SECTION 6

IEC61131-3 PROGRAMMING

SEQUENCE FUNCTION CHARTS
CONTENTS

6.1 CONCEPTS.......................................................................................................................................................................................................................................................................3
  6.1.1 CHART STRUCTURE................................................................................................................................................................................................................3
  6.1.2 SIMULTANEOUS SEQUENCES .........................................................................................................................................................................................5

6.2 METHODOLOGY.........................................................................................................................................................................................................................6
  6.2.1 STEPS.................................................................................................................................................................................................................................6
  6.2.2 TRANSITIONS........................................................................................................................................................................................................7
  6.2.3 ACTIONS ..................................................................................................................................................................................................................10
  6.2.4 ACTION QUALIFIERS...............................................................................................................................................................................................13

6.3 SFC EXECUTION........................................................................................................................................................................................................18
  6.3.1 SFC EVALUATION RULES.....................................................................................................................................................................................18
  6.3.2 SFC RULES OF EVOLUTION ...........................................................................................................................................................................19
  6.3.3 SAFE AND UNSAFE DESIGN ...........................................................................................................................................................................21

6.4 PROCEDURES........................................................................................................................................................................................................23
  ❑ Procedure 1 – Create an SFC Chart: ......................................................................................................................................................................................23
  ❑ Procedure 2 – Adding another step to the SFC Chart: ................................................................................................................................................23
  ❑ Procedure 3 – Adding an action to a step: ............................................................................................................................................................24
  ❑ Procedure 4a – Inserting an SFC divergent branch: ........................................................................................................................................24
  ❑ Procedure 4b – Inserting an SFC simultaneous branch: ....................................................................................................................................25
  ❑ Procedure 5a – Inserting an alternative divergence: .......................................................................................................................................25
  ❑ Procedure 5b – Inserting a simultaneous divergence: .......................................................................................................................................26
  ❑ Procedure 6a – Step Properties: ...........................................................................................................................................................................26
  ❑ Procedure 6b – Action and Transition Properties: ..........................................................................................................................................27
6.1 Concepts

6.1.1 Chart structure

A sequence is depicted as:

- A series of steps shown in rectangular boxes connected by vertical lines
- each step represents a particular state of the system being controlled
- each step can be associated with one or more actions
- each connecting line has a horizontal bar representing a transition
- a transition can be described using ST, FBD or LD
- a transition is associated with a condition which, when true,
  - causes the step before the transition to be deactivated
  - and the step that follows the transition to be activated
- The flow of control is generally down the page but branches can be used to go back to earlier steps

Alternative steps in a sequence can be selected:

- using a divergence construct where more than one transition is associated with leaving a step
- a divergence sequence can rejoin another sequence using a convergent path
- any number of divergent and convergent paths can be used enabling complex behavior to be depicted
- using a divergent path, only one step in one of the alternative sequences can be active at any time
Example 1 - SFC:

Main phases of a process controlling a reactor vessel

1. Start step
   Initial step concerned with holding the vessel in a ready state waiting for the Start switch to be set ON

2. Fill step
   Becomes active when StartSwitch is set to 'ON'. As the 'Fill' step is activated, the 'Start' step is deactivated in the same instant

3. Stir step
   Is activated when the 'VesselFull signal is detected. Duration is timed using an elapse timer FB Timer1.

4. Drain step
   Is activated on Timer1 time elapsing.

5. Stop step
   Is reached when the StartSwitch is placed in the 'Off' position.

Example 2 - Divergent and convergent paths
6.1.2 Simultaneous sequences

A construct known as a simultaneous sequence divergence makes it possible to activate parallel sequences.

Parallel sequences are useful in many batch processing applications where the main sequence can be used for sequencing the primary process phases, while secondary parallel sequences can be monitoring the operating constraints of the process.

*Example:* check that plant temperatures and pressures are within operating limits, and, if not, initiate a shut-down sequence
6.2 Methodology

6.2.1 Steps

Each step should have a unique name and should only appear once in a SFC.

Step and transition names are local to the POU, in which the SFC exists.

The behavior of any step can be described by associating one or more action blocks with the step. Each action block can be described using any of the IEC languages.

There are two forms of step:

- **Normal steps** - are depicted in rectangular boxes with the step name in the centre.

- **Initial steps** - These are depicted in identical rectangular boxes but with vertical bars. Only one initial step should be specified for any SFC as it defines the step that will be activated on PLC cold-start.

**Step executing and time variables**

Every step is associated with two variables that can be used within the rest of the SFC to synchronize and to monitor step activation.

1. **The step active flag (.X)** - is a boolean variable that is only set true while a particular step is active can be tested within other parts of an SFC, to check whether a particular step is active.

   Form `<StepName>.X`

   This flag can be provided as a graphical output from the step box. For example, in the above example, if the transition condition between TempChk and PressChk is `Drain.X = 1`, it will wait until the Fill step is active before continuing to the PressChk step.

2. **The Elapse timer (.T)** - is a variable of data type TIME. When a step is first activated, the elapse time variable is set to zero. While a step is active, it is updated to indicate how long the step has been active. When the step ceases to be active, the .T variable retains the elapse time.

   Form `<StepName>.T`
6.2.2 Transitions

Different constructs provided for describing transition conditions using text or graphics:

**Structured Text**

Any ST expression that results in a boolean result can be used to describe a transition condition. (note that the ST is embedded in an ST worksheet that is lined to this transition)

**Ladder diagram**

A ladder rung can be used as a transition condition. When the rung provides a power flow, the transition condition becomes true

**FB diagram**

Any FBD network that generates a boolean output can be connected to a transition
Transition connector

A transition connector can be used in situations where:
The transition condition is shown in another part of a SFC

Named Transition

Transitions can be identified by name
The definition of the named transition can then be defined on another diagram or page using ST, FBD, LD or IL

Named Transition Definition using ST

A named transition condition can be described using a ST expression. The expression should only return a boolean value, and this value is loaded into the name of the transition (e.g. Trans1):

\[
\text{Trans1: } = \text{AB1 AND CX3 OR CX5 AND (TX3} \geq 100.2)\];
Named Transition definition using FBD

TRANSITION TRANSGO:
IN1
  AND
Enable1
OR TransGo
IN2
  AND
Enable2
END_TRANSITION

The final output of the FBD network provides the value of the transition

Named Transition Definition using LD

TRANSITION TRANS21:
%IX21 EX10 Trans21

END_TRANSITION

The coil, which is set by the power flow in the rung, has the same name as the transition. When the coil is set, the transition condition is true.

Named Transition Definition using IL

TRANSITION Trans21 :
LD %IX21
AND EX10
AND FDIR21
END_TRANSITION

The value of ‘result register’ after executing the IL instructions, defines the value of the transition condition. When the value, that should be boolean data type, is 1 the transition condition is true.
6.2.3 Actions

Every step can be associated with one or more actions:

- An action contains a description of some behavior that should occur as a result of the step being activated
- An action is depicted as a rectangular box that is attached to a step
- The description of an action can be in ST, FBD, LD, IL, or even another SFC.
- Every action has a unique name in the current POU

Main features of actions

```
Start
001 Start_Switch
Fill
N Prompt_Operator
Action Qualifier
N Start_Pump
P Open_Valves
```
Action defined using LD

**Pump_Control**
The rungs are enclosed in a rectangular box, with the name of the action placed at the top

```
X10 AL1 AL2 Pump1
AL3 Pump1 AuxX1
```

Action defined using FBD

**Stir_Control**
The FBD network is enclosed in a rectangle box, with the name of the action placed on top
If an action is configured to execute continually it can be used to contain feedback control loops

```
1000 SP OUT StirDrive
1500 GT OverSpeed
```

Action defined using ST

A variable changed by an ST statement can be used as an action indicator variable

```
ACTION OpenValves :
    IF BatchType = “AB0_100” THEN
        ValveAB := OPEN;
    ELSE
        ValveAC := OPEN;
    END_IF;
    DrainValve := CLOSE;
    Vent := OPEN;
END_ACTION
```
Action defined using IL

A variable changed by an IL instructions can be used as an action indicator variable.

```
ACTION AddX2 :
    LD 1
    ST TankX2.Enable
    LD 100
    ST TankX2.ShotLevel
    CAL TankX2
    S X2_Added
END_ACTION
```

Action defined using an SFC

It is possible to define an action using another SFC. In this way, complex, hierarchical SFCs can be built.

(no example shown)
6.2.4 Action qualifiers

By default, actions execute continually while their associated steps are active. Such actions have the ‘N’ qualifier, which is placed in the first field of the action block.

<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Non-stored, default, same as ‘N’</td>
</tr>
<tr>
<td>N</td>
<td>Non-stored, executes while associated step is active</td>
</tr>
<tr>
<td>R</td>
<td>Resets a stored action</td>
</tr>
<tr>
<td>S</td>
<td>Sets an action active, i.e. stored</td>
</tr>
<tr>
<td>L *</td>
<td>Time limited action, terminates after a given period</td>
</tr>
<tr>
<td>D *</td>
<td>Time delayed action, starts after a given period</td>
</tr>
<tr>
<td>P</td>
<td>A pulse action that only executes once when a step is activated, and once when a step is deactivated</td>
</tr>
<tr>
<td>SD *</td>
<td>Stored and time delayed. The action is set active after a given period, even if the associated step is deactivated before the delay period</td>
</tr>
<tr>
<td>DS *</td>
<td>Action is time delayed and stored. If the associated step is deactivated before the delay period, the action is \textbf{not} stored</td>
</tr>
<tr>
<td>SL *</td>
<td>Stored and time limited. The action is started and executes for a given period</td>
</tr>
</tbody>
</table>

(Note: * These qualifiers all require a time period)
**N qualifier**

The action ‘Action1’ executes continually while ‘StepA’ is active, i.e. while Step1.X = 1. It executes one extra time after the step is deactivated.

**S (Set) and R (Reset) qualifiers**

The S qualifier causes the action ‘Action1’ to execute when the step becomes active. The action continues to execute until an action block with an R qualifier that reference ‘Action1’ is reached. It ceases action one cycle after ‘Stepn’ is activated.
**L time limited qualifier**

The action executes when the step becomes active. It ceases to become active after the given time period defined in the qualifier, or when the step ceases execution, whichever comes first.

```
L T#4s  Action1
```

**D time delayed qualifier**

The action executes a specified time period after the step becomes active. If the step deactivates before the delay period has expired then the action does not execute.

```
D T#4s  Action1
```
**P pulse qualifier**

The action executes once when the step is activated. **Note that like all action blocks, the action executes once when de-activated.** Therefore care should be used with this qualifier as the action is executed twice.

![Diagram of P pulse qualifier]

**SD stored and time delayed qualifier**

This is a combination of the ‘S’ and the ‘D’ qualifiers. Action1 is executed after a delay time period, and continues to execute until an ‘R’ qualifier is reached. Action1 will execute even if Step1 deactivates before the delay period expires.

![Diagram of SD stored and time delayed qualifier]
DS time delayed and stored qualifier

Similar to the **SD** qualifier except that if the step deactivates the action will not execute.

![Diagram of DS time delayed and stored qualifier]

SL stored and time limited qualifier

A combination of the S (set) and the L (time limited) qualifiers. The action executes and continues to execute until the specified time period expires, or an R qualifier is executed, whichever comes first.

![Diagram of SL stored and time limited qualifier]
6.3 SFC execution

6.3.1 SFC Evaluation Rules

- The initial step of an SFC is always activated when the POU executes after system initialization
- actions associated with the initial step are executed.

- At the start of each evaluation the current set of active steps is determined. All transitions that are associated with the active steps are evaluated.

- Actions which nominally ceased execution in the previous SFC evaluation are executed one last time.

- All actions which are active are executed.

- When transition conditions become true any active steps that precede it are deactivated and their succeeding steps are activated.

- Any actions, having an execution enabling condition that ceases, are marked as inactive.
6.3.2 SFC rules of evolution

The SFC rules of evolution are network design guidelines to ensure that behavior is unambiguous:

- Two steps can never be directly linked; they should always be separated by one transition.

- Two transitions can never be directly linked; they should always be separated by a step.

- If a transition from one step leads to two or more steps, then such steps indicate simultaneous sequences. Simultaneous sequences continue independently.

- When designing an SFC, the time to clear a transition, deactivate preceding steps and activate succeeding steps may be regarded as occurring instantaneously.

- Where more than one transition condition is true at the same time, all the deactivations and activations of associated steps appear to occur in zero time.

- A transition condition of a step is not evaluated until all the behavior resulting from the active step has propagated through the POU. For example, if a step is activated that has a transition condition that is always true, all applicable actions are executed once before the step is deactivated.

In addition to these rules, there are rules of evolution that apply to how branching and looping can be defined.
Evolution of Divergent Paths

**Order of Precedence** - the order of evaluation is left to right. If an SFC branches to two or more steps, and their transitions are TRUE at the same time, then the left step will be executed.

**Sequence Skip/Loop** – a divergent path with no steps. A divergent path with no steps, but with one transition, can be defined, and may flow in either direction up (loop) or down (skip) the chart.

Convergent simultaneous sequences - A number of simultaneous sequences can converge to a single transition (rendezvous point) and then diverge to a number of simultaneous sequences. However all simultaneous sequences must be complete before the single transition condition is evaluated.
6.3.3 Safe and Unsafe Design

Certain arrangements of branches in SFC networks can lead to unsafe designs.

The following is an example of an unsafe SFC design. In this case, if the divergent path to ‘StepD’ is taken, the sequence returns to ‘Step1’. However, one of the steps in the ‘Step2’ or ‘Step4’ sub-sequences could still be active, and the sequence could loop back into these sub-sequences causing unpredictable behavior.
The following is an example of an **unreachable** SFC. Remember that a simultaneous sequence convergence requires that all preceding steps are executed. But StepC and StepD are mutually exclusive. Therefore Transition TrX will never be evaluated and StepX will never executed. In other words, the SFC will hang at this point.

The alternative SFC on the right shows the two divergent steps (StepC and StepD) converging into StepE (an empty step), so that the simultaneous sequence convergence will activate when both Step4 and StepE are executed.
6.4 Procedures

SFCs can be created in a POU with SFC properties. The Logic Designer has an SFC creation toolbar:

Procedure 1 – Create an SFC Chart:

1. Click on the worksheet.
2. Click on the ‘Create Sequence’ button on the toolbar. The following chart appears:

Procedure 2 – Adding another step to the SFC Chart:

1. Click on the step or transition in the SFC chart. If there is more than one step, click on the one before which the new step will be placed.
2. Click on the ‘Create Sequence’ button in the toolbar. The new step appears after the selected step/transition.
Procedure 3 – Adding an action to a step:

1. Click on the first action within the step.
2. Click on the ‘Create Action’ button. A new action is created.

Procedure 4a – Inserting an SFC divergent branch:

1. Click on the SFC Branch button.
2. Click on or below a step. An ‘X’ will appear on the vertical line below the step, and a horizontal line with transition can be dragged to any position.
3. Click to place the transition.
4. Click on a step or transition.
   - If a step is clicked on, the line will connect directly above the step.
   - If a transition is clicked on, the line will connect directly below the transition.

A step can be added to the divergent path by using Procedure 2 above.
Procedure 4b – Inserting an SFC simultaneous branch:

1. Click on the SFC Branch button.
2. Click on or below a transition. A set of parallel lines will appear, and a step.
3. Click to place the step.
4. Click on a step or transition.
   - If a step is clicked on, the lines will connect directly below the step.
   - If a transition is clicked on, the lines will connect directly above the transition.

Procedure 5a – Inserting an alternative divergence:

1. Click on a step.
2. Click on the Simultaneous/Alternative Divergence button.
3. Select the number of paths in the dialog box. Click OK.
4. The divergent branch will appear directly below the step.
Procedure 5b – Inserting a simultaneous divergence:

1. Click on a transition.
2. Click on the Simultaneous/Alternative Divergence button.
3. Select the number of paths in the dialog box. Click OK.
4. The divergent branch will appear directly below the transition.

Procedure 6a – Step Properties:

To access Step properties, right click on the step and select Object Properties. The following dialog appears:
Procedure 6b – Action and Transition Properties:

Right click on the action or transition. From the context menu, the following two selections access certain properties of the action or transition:

- **Object Open** – Selects the language to be used in the transition or action.
- **Object Properties** – If transition, select name and whether it is detail or direct connection.
  - If Action, select name and qualifier.
APPENDIX 1

PAS POU FUNCTIONS
CONTENTS

Appendix 1 – PASPOU Function Block Details.................................................................3
  A1.1 Input Output Blocks ..............................................................................................3
  A1.2 Controller Blocks .................................................................................................4
  A1.3 Manual Loader Blocks ........................................................................................5
  A1.4 Signal Setter Blocks ...........................................................................................6
  A1.5 Signal Limiter Block ...........................................................................................7
  A1.6 Signal Selector Blocks .........................................................................................7
  A1.7 Signal Distributor Blocks .....................................................................................8
  A1.8 Calculation Blocks ..............................................................................................9
  A1.9 Sequence Auxiliary Blocks .................................................................................13
  A1.10 Switch Instrument and Motor Control Blocks ..................................................15
    A1.10-1 Operation of the Switch Instrument ...............................................................16
    A1.10-2 Answerback Input Function .........................................................................17
    A1.10-3 Output Signal Conversion Function ..............................................................17
    A1.10-4 Answerback Alarm Function .........................................................................18
    A1.10-5 Other Switch Instrument Functions ..............................................................18
Appendix 1 – PASPOU Function Block Details

A1.1 Input Output Blocks

<table>
<thead>
<tr>
<th>Block</th>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analog Input</td>
<td>PAS_AI_ANLG</td>
<td>Analog input block</td>
</tr>
<tr>
<td>Output</td>
<td>PAS_AI_PCNT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PAS_AI_PULS_CI</td>
<td>Pulse Input Block – Pulse Counter</td>
</tr>
<tr>
<td></td>
<td>PAS_AI_PULS_QT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PAS_AI_TEMP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PAS_AO_ANLG</td>
<td>Analog output block</td>
</tr>
<tr>
<td></td>
<td>PAS_PVI</td>
<td>Input Indicator</td>
</tr>
<tr>
<td>Digital Input</td>
<td>PAS_DI_PUSHB</td>
<td>Digital input – pushbutton type</td>
</tr>
<tr>
<td>Output</td>
<td>PAS_DI_STS</td>
<td>Digital input – status type</td>
</tr>
<tr>
<td></td>
<td>PAS_DO_STS</td>
<td>Digital output – status type</td>
</tr>
<tr>
<td></td>
<td>PAS_DO_STS_PW</td>
<td>Digital output – pulse width type</td>
</tr>
</tbody>
</table>

![Block Diagrams]

---

**Note:** The images of the block diagrams are not provided in the text. They are likely visual representations of the function blocks mentioned in the table.
A1.2 Controller Blocks

<table>
<thead>
<tr>
<th>Block</th>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controllers</td>
<td>PAS_PID</td>
<td>PID controller block</td>
</tr>
<tr>
<td></td>
<td>PAS_PI-HLD</td>
<td>Sampling PI controller block</td>
</tr>
<tr>
<td></td>
<td>PAS_ONOFF</td>
<td>2-position ON/OFF controller block</td>
</tr>
<tr>
<td></td>
<td>PAS_ONOFF_G</td>
<td>3-position ON/OFF controller block</td>
</tr>
</tbody>
</table>

![Controller Blocks Diagram]
## A1.3 Manual Loader Blocks

<table>
<thead>
<tr>
<th>Block</th>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual Loader</td>
<td>PAS_MLD</td>
<td>Manual loader block</td>
</tr>
<tr>
<td></td>
<td>PAS_MLD_BT</td>
<td>Manual loader block with Auto/Man switch</td>
</tr>
<tr>
<td></td>
<td>PAS_MLD_PB</td>
<td>Manual loader block with Auto/Man switch</td>
</tr>
</tbody>
</table>

![Diagram of PAS_MLD block](image1)

![Diagram of PAS_MLD_BT block](image2)

![Diagram of PAS_MLD_PB block](image3)
## A1.4 Signal Setter Blocks

<table>
<thead>
<tr>
<th>Block type</th>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal Setter</td>
<td>PAS_RATIO</td>
<td>Ratio set block</td>
</tr>
<tr>
<td></td>
<td>PAS_RATIO_RT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PAS_PG_L30</td>
<td>30-zone program set block</td>
</tr>
<tr>
<td></td>
<td>PAS_PG_L30_BP</td>
<td></td>
</tr>
</tbody>
</table>

### PAS_RATIO
- `IN` IN
- `Out` OUT
- `PLS_IN` PLS_IN
- `CAS_IN` CAS_IN
- `TSI` TSI
- `INT` INT
- `PRM_IN` PRM_IN
- `PRM_OUT` PRM_OUT

### PAS_PG_L30
- `IN` IN
- `PRG_RST` PRG_RST
- `PRG_END` PRG_END
- `INTLOCK` INTLOCK
- `ENG_PRM` ENG_PRM
- `PRM_IN` PRM_IN
- `PRM_OUT` PRM_OUT
A1.5 Signal Limiter Block

<table>
<thead>
<tr>
<th>Block type</th>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal Limiters</td>
<td>PAS_VELLIM</td>
<td>Velocity limiter block</td>
</tr>
<tr>
<td></td>
<td>PAS_VELLIM_PB</td>
<td></td>
</tr>
</tbody>
</table>

A1.6 Signal Selector Blocks

<table>
<thead>
<tr>
<th>Block type</th>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal Selectors</td>
<td>PAS_AS_H</td>
<td>High auto-selector block</td>
</tr>
<tr>
<td></td>
<td>PAS_AS_M</td>
<td>Medium auto-selector block</td>
</tr>
<tr>
<td></td>
<td>PAS_AS_L</td>
<td>Low auto-selector block</td>
</tr>
</tbody>
</table>
A1.7 Signal Distributor Blocks

<table>
<thead>
<tr>
<th>Block type</th>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal Distributors</td>
<td>PAS_FOUT</td>
<td>Cascade control signal distribution block</td>
</tr>
<tr>
<td></td>
<td>PAS_FFSUM</td>
<td>Feedforward control signal addition block</td>
</tr>
<tr>
<td></td>
<td>PAS_FFSUM_BL</td>
<td></td>
</tr>
</tbody>
</table>

- **PAS_FOUT**
  - Block type: Cascade control signal distribution block
  - Input: RB_IN1, RB_IN2, RB_IN3, RB_IN4, RB_IN5, RB_IN6, RB_IN7, RB_IN8, CAS_IN
  - Output: PRM_OUT

- **PAS_FFSUM**
  - Block type: Feedforward control signal addition block
  - Input: RB_IN, IN, CAS_IN, BIN, TSI, TIN, INTLOCK, ENG_PRM, PRM_WT
  - Output: PRM_WT
## A1.8 Calculation Blocks

<table>
<thead>
<tr>
<th>Block Type</th>
<th>Code</th>
<th>Block Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analog Calculation</td>
<td>PAS_LDLAG</td>
<td>Lead/Lag</td>
</tr>
<tr>
<td></td>
<td>PAS_DLAY</td>
<td>Dead Time</td>
</tr>
<tr>
<td></td>
<td>PAS_AVE_M</td>
<td>Moving Average</td>
</tr>
<tr>
<td></td>
<td>PAS_AVE_C</td>
<td>Cumulative Average</td>
</tr>
<tr>
<td></td>
<td>FUNC-VAR</td>
<td>Variable line-segment function</td>
</tr>
<tr>
<td></td>
<td>PAS_P_CFL</td>
<td>Pressure Compensation of gas flow</td>
</tr>
<tr>
<td></td>
<td>PAS_T_CFL</td>
<td>Temperature Compensation of gas flow</td>
</tr>
<tr>
<td></td>
<td>PAS_TP_CFL</td>
<td>Pressure &amp; Temp Compensation of gas flow</td>
</tr>
<tr>
<td></td>
<td>PAS_ASTM1</td>
<td>Oil Temp Correction: Old JIS</td>
</tr>
<tr>
<td></td>
<td>PAS_ASTM2</td>
<td>Oil Temp Correction: New JIS</td>
</tr>
<tr>
<td></td>
<td>PAS_SW-13</td>
<td>3-pole 1-position Selector Switch</td>
</tr>
<tr>
<td></td>
<td>PAS_SW-31</td>
<td>1-pole 3-position Selector Switch</td>
</tr>
<tr>
<td></td>
<td>PAS_SW-19</td>
<td>9-pole 1-position Selector Switch</td>
</tr>
<tr>
<td></td>
<td>PAS_SW-91</td>
<td>1-pole 9-position Selector Switch</td>
</tr>
<tr>
<td></td>
<td>PAS_BD_BUF_R</td>
<td>Batch data set block</td>
</tr>
<tr>
<td></td>
<td>PAS_BD_BUF_T</td>
<td>Batch data set block</td>
</tr>
</tbody>
</table>

### Lead/Lag (LDLAG)

```
PAS_LDLAG 1
- RB_IN  RB_OUT
- IN    OUT
- ENG_PRM PRM_RD_OUT
- PRM_WT  PRM_WT
```

### Delay (DLAY)

```
PAS_DLAY 1
- RB_IN  RB_OUT
- IN    OUT
- ENG_PRM PRM_RD_OUT
- PRM_WT  PRM_WT
```

---

TE 33AU2C1-01  
*Appendix 1 – PAS POU Functions*  
A1 - 9
Moving Average (AVE-M)


diagram

Cumulative Average (AVE-C)


diagram

Variable line-segment function (FUNC-VAR)


diagram

\[ CPV = GAIN \times \frac{\text{SUM of Input}}{\text{Number of Inputs}} \]
Pressure & Temp Comp. of Gas Flow (PAS_TP_CFL, PAS_P_CFL, PAS_T_CFL)

Oil Temp Correction: Old JIS (PAS_ASTM1), New JIS (PAS_ASTM2)

Selector Switch (PAS_SW-31, PAS_SW-91)
Appendix 1 – PAS POU Functions

Distributor Switch (PAS_SW-13, PAS_SW-19)

Batch Data Set Blocks (PAS_BD_BUF_R, PAS_BD_BUF_T)
### A1.9 Sequence Auxiliary Blocks

<table>
<thead>
<tr>
<th>Block Type</th>
<th>Code</th>
<th>Block Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timers</td>
<td>PAS_TN</td>
<td>Timer</td>
</tr>
<tr>
<td>Switch Instrument Blocks</td>
<td>SI-1</td>
<td>Switch Instrument Block – 1 input</td>
</tr>
<tr>
<td></td>
<td>SI-2</td>
<td>Switch Instrument Block – 2 inputs</td>
</tr>
<tr>
<td></td>
<td>SO-1</td>
<td>Switch Instrument Block – 1 output</td>
</tr>
<tr>
<td></td>
<td>SO-2</td>
<td>Switch Instrument Block – 2 outputs</td>
</tr>
<tr>
<td></td>
<td>SIO-11</td>
<td>Switch Instrument Block – 1 input/1 output</td>
</tr>
<tr>
<td></td>
<td>SIO-21</td>
<td>Switch Instrument Block – 2 input/1 output</td>
</tr>
<tr>
<td></td>
<td>SIO-12</td>
<td>Switch Instrument Block – 1 input/2 output</td>
</tr>
<tr>
<td></td>
<td>SIO-22</td>
<td>Switch Instrument Block – 2 input/2 output</td>
</tr>
</tbody>
</table>

**Timer Block (TM)**

![Timer Block Diagram](image)

- **PV Value**: Present Value
- **PV**: Process Variable
- **NH**: Normal High
- **IH**: Intermediate High
- **LV**: Low Variable
- **DH**: Deadband High
- **DL**: Deadband Low
- **TIME**: Time
- **ACTION**: Action
- **RUN**: Run
- **STOP**: Stop
- **NR**: Normal Running
- **P/A**: P/A
- **CT**: Control
- **END**: End

The timer block diagram shows the status transitions between different operational modes, indicating how the timer behaves under various conditions.
Counter Block (CT)
A1.10 Switch Instrument and Motor Control Blocks

Switch Instrument blocks are function blocks that provide an operator and sequence interface to valves, motors, pumps and other actuating devices. They provide facilities to allow a sequence table or logic chart to control the field device easily.

There are several types of Switch Instrument Blocks depending on the number and types of inputs/outputs to the blocks:

<table>
<thead>
<tr>
<th>Block Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SI-1</td>
<td>Switch Instrument with 1 input</td>
</tr>
<tr>
<td>SI-2</td>
<td>Switch Instrument with 2 inputs</td>
</tr>
<tr>
<td>SO-1</td>
<td>Switch Instrument with 1 output</td>
</tr>
<tr>
<td>SO-2</td>
<td>Switch Instrument with 2 outputs</td>
</tr>
<tr>
<td>SIO-11</td>
<td>Switch Instrument with 1 input and 1 output</td>
</tr>
<tr>
<td>SIO-21</td>
<td>Switch Instrument with 2 inputs and 1 output</td>
</tr>
<tr>
<td>SIO-12</td>
<td>Switch Instrument with 1 input and 2 outputs</td>
</tr>
<tr>
<td>SIO-22</td>
<td>Switch Instrument with 2 inputs and 2 outputs</td>
</tr>
<tr>
<td>SIO-12P</td>
<td>Switch Instrument with 1 input and 2 pulse outputs</td>
</tr>
<tr>
<td>SIO-22P</td>
<td>Switch Instrument with 2 inputs and 2 pulse outputs</td>
</tr>
</tbody>
</table>
A1.10-1  Operation of the Switch Instrument

The digital input(s) are connected to the IN terminals and are converted into a PV (0,1,2) through the Answerback Input Function. The OUT terminal connects to 1 or 2 digital outputs and is driven by the MV (0,1,2) through the Output Signal Conversion Function. An Answerback Check Function checks that the input (PV) corresponds to the output command (MV) after a set time. If not, an Answerback Alarm is generated.

When in manual, the MV is set by the operator clicking on the pushbuttons on the faceplate. When in auto, the MV is written to by the CSV which in turn is set by a sequence table or logic chart.
A1.10-2 Answerback Input Function

The digital input status is converted to an integer in the PV as follows:

<table>
<thead>
<tr>
<th>Number of Inputs</th>
<th>Input Status</th>
<th>PV</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>n</td>
<td>n+1</td>
</tr>
<tr>
<td>ON</td>
<td>2</td>
<td>OFF</td>
</tr>
<tr>
<td>OFF</td>
<td>0</td>
<td>ON</td>
</tr>
</tbody>
</table>

These relate to the function block pushbuttons as follows:

- MV=2
- PV=2

- MV=1
- PV=1

- MV=0
- PV=0

A1.10-3 Output Signal Conversion Function

The digital output status is converted from an integer (MV) as follows:

<table>
<thead>
<tr>
<th>Number of Outputs</th>
<th>MV</th>
<th>Output Status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n</td>
<td>n+1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>ON</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>OFF</td>
</tr>
</tbody>
</table>

| 2 (pulse type) | 2  | ON | OFF |
| 0              | OFF| ON |

These relate to the function block pushbuttons as shown above.
A1.10-4 Answerback Alarm Function

When the output of the Switch Instrument block is changed (i.e. the MV is changed), a masking timer is started. If the timer times out before the input matches the output (i.e. PV ≠ MV) then an Answerback alarm is given.

<table>
<thead>
<tr>
<th>Discrepancy</th>
<th>Alarm Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>MV &gt; PV</td>
<td>ANS+</td>
</tr>
<tr>
<td>MV &lt; PV</td>
<td>ANS-</td>
</tr>
</tbody>
</table>

In other words, if a request to open a valve or start a pump is issued (i.e. MV = 2) and the valve fails to open or the pump fails to start (i.e., PV ≠ 2), then the ANS+ alarm status is generated.

The masking timer setpoint (MTM) is set through the tuning panel.

A1.10-5 Other Switch Instrument Functions

- **Tracking Function**
  
  If the TSW switch is on, then the output tracks the input, i.e., MV = PV. The TSW can be set from a sequence, or through a connection to the TSI terminal.

- **Answerback Bypass**
  
  If the BPSW switch is on, then the input is forced to the output, i.e., PV = MV, and no answerback alarm check takes place. The BPSW can be set from a sequence, or through the SWI terminal.

- **Mode Change Interlock Function**
  
  When the interlock input is on, then the Switch Instrument is forced to Manual and cannot be set to Auto. The interlock input is a terminal (INTLOCK) to which a switch or digital input is connected.
SECTION 2

FCX FUNCTIONS

VDS Versatile Data Server Software

Application Portfolio

Stardem™

Autonomous Controllers

FCN

FCH
2.6.5 Converting Between Data Types..........................................................................................76
  2.6.5.1 Connecting between IEC Blocks and PAS_POU blocks.................................................76
  2.6.5.2 Converting Boolean switches to PAS_POU switches.....................................................78
  2.6.5.3 Converting General Data Types....................................................................................79
2.6.6 Cascade Read-Back Connections.......................................................................................80
  2.6.6.1 Cascade Control Loops ................................................................................................80
  2.6.6.2 Analog Outputs .............................................................................................................83
2.6.7 Reporting Alarms to the HMI .........................................................................................84
2.6.8 Inter-FCX Communication (SD_FCXCOM_LIB)..............................................................85
  2.6.8.1 Unconfirmed Communications .....................................................................................87
  2.6.8.2 Confirmed Communications .......................................................................................89
  2.6.7.3 Programming FCX Read/Write functions in Logic Designer .......................................92
2.7 Debugging Tools..................................................................................................................93
  2.7.1 Running Debug Mode: .....................................................................................................93
  2.7.2 Softwiring Procedure........................................................................................................94
  2.7.3 Logic Analyzer Procedures .............................................................................................96
  2.7.4 The Watch Window .........................................................................................................97
2.8 HTML Applications ............................................................................................................99
2.1 Introduction

2.1.1 FCX System Configuration

The FCN and FCJ, hereafter referred to as FCX, are based on the Intel Pentium P166 CPU.

The controller functions run in the CPU using an IEC61131 compliant PLC engine. This in turn runs on the real-time, multitasking operating system, VxWorks. See section 2.2.3 for more information about the multi-tasking capabilities.
2.1.2 FCX Programming

The FCX is programmed using the IEC61131-3 programming standard. This standard specifies five languages as follows:

- Function Blocks (FB)
- Logic Diagrams (LD)
- Structured Text (ST)
- Sequence Function Charts (SFC)
- Instruction List (IL)

The standard defines a small set of basic functions which have been extended by Yokogawa to include a set of high level functions similar to the CS3000 (DCS) Function Block set. A description of these blocks are included in this section.

As well as defining the programming languages, the standard also defines the data structure for all data that is defined in the system. Data can be associated with function blocks, and therefore tags, or be designated as global or external variables, that is data that is available to the system as a whole. This includes I/O which are defined as global variables.

The programming for the system is carried out through the Logic Designer. Through this, a project is created that contains the configuration for the system. Within the project, the programs are created and linked to the FCXs through tasks, which define how the programs operate within the FCX.

The hardware set up is carried out through the Resource Configurator. This configures the CPU and I/O of the FCX, and allows upload and storage of FCX data on the PC.
2.1.3 Project Layout

A project resides in three different locations:

- FCX Builder – edited by Logic Designer
- Dataserver Builder – edited by Object builder and Graphic builder
- Webserver – created by HMI Deployment tool

To reduce the amount of engineering, each of these builders has an import/export tool:

**Resource Configurator**
- Plug & Play Function (reads setup on FCX)
- There is no import/export facility, but can copy and paste from/into Excel

**Logic Designer**
- Cannot import configuration from Resource Configuration
- Import/Export Device Label Definition from other project
- Import/Export CSV files in general

**Object Builder**
- Import definition from Logic Designer (ADLST)
  - only for PAS POU modules
- Object building function for other modules.

**Graphic Designer**
- Import/Export function
- Browser for drag & drop of tag names in graphic builder.
2.1.4 Application Portfolios and the IEC Layering Concept

Because of the layering functionality, and the security function, Application Portfolios can be created by the user that protects their intellectual property.

Layering means that functions and function blocks can be subsets of higher level functions and function blocks with as many layers as is required. See section 2.3.2 for more information on how layering is achieved.

A function block which may contain many other functions and function blocks can be password protected so that no-one may view or edit the program inside it. See section 2.5.3 for more information on security.
2.2 FCX Hardware Configuration

2.2.1 I/O Interface

The I/O interface provides a connection between the control software and the process I/O. The process I/O can be in the form of:

- On-board hardware I/O
- Fieldbus I/O
- PLC communications

I/O is configured directly using the **Resource Configurator** (see section 2.2.5). This allows the configuration of **device label** names and other I/O parameters.

This I/O configuration has to be connected to the control software so that the programs can read and write to the external I/O, as can be seen in the diagram below. This is done in the **Physical Hardware** folder of the project (in **Logic Designer**) in the **Device Label Definition** file.
Device Label Definition

The I/O tags, or device labels, as defined through the Resource Configurator must be entered into the DeviceLabelDefinition file. This provides the direct linking between the process I/O points and the software. When the system is built and loaded, the label names here are matched to the label names of the I/O in the hardware to allow the transfer of data between the software and the hardware.

To open the Device Label Definition, double click on the DeviceLabelDefinition file in the FCX Folder in Logic Designer:

- The Device label name represents an input with the same name in the hardware definition. Scaling and other parameters can be defined.

- The Reading and Write buttons allow for the importing and exporting of previously setup configurations, allowing for the copying of generic I/O assignments to other projects or resources.

- The IO Group button allows changing the configuration of the IO Groups, but this does not need to be changed, and therefore information in this window should not be modified.

- Sharing of I/O by different tasks of different scan cycles is not recommended, particularly in the case of outputs and in dual-redundant CPU configurations.
2.2.2 Memory Structure of the FCX

There are three memory locations in the FCX:

- **Main Memory** – DRAM (32 MB), volatile, holds the running program which is loaded on boot-up from the boot project in the Flash RAM. Also hold current data (tuning parameters). Note that all this data is lost when power is cycled.

- **Data Memory** – SRAM (512KB), battery backed RAM on motherboard. Holds tuning parameters, which can be backed-up to Flash RAM. It is also known as non-volatile RAM (NVRAM).

- **Flash RAM** (plug-in Compact Flash card) – holds system software, boot project and project source file, tuning parameters (retained variables).
Retention of data (Tuning Parameters):

Both Global and Local variables can be retained in the event of loss of power to the FCX. Such variables are known as “retained data”. This includes access parameters (tuning parameters) of PAS POU blocks. Data can be retained in any of the above mentioned memory areas, and this is set as follows:

- Check the “Retain” box (in Logic Designer – see 2.4.4) for the required variables.
  1. If “enable Hard-backup for retained data” is checked in the Resource Configurator then data is retained in Data Memory (non-volatile memory)
  2. If “enable Hard-backup for retained data” is not checked then data is retained in Main Memory (volatile memory). This allows faster access to data but is less secure.

- Data can also be backed-up to Flash Ram:
  1. Initiated by a global variable (GS_RETAIN_SAVE_SW), a one shot process. A button on a graphic can be linked to this variable to give simple user access to this function.
  2. This data is restored to the program if data has been lost from the Data memory or Main memory

Initial Values:

Both Global and Local variables can have initial values. These are specified in the variable setup in Logic Designer in the “Init” field. These are stored in the Flash RAM and are loaded into the Main RAM on power-up or Cold Start of the FCX.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Usage</th>
<th>Description</th>
<th>Address</th>
<th>Init</th>
<th>Retain</th>
<th>F...</th>
<th>OPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1INI</td>
<td>BOOL</td>
<td>VAR_INPUT</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1MODE</td>
<td>BOOL</td>
<td>VAR_INPUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUSTAT</td>
<td>BOOL</td>
<td>VAR</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1DTY</td>
<td>BOOL</td>
<td>VAR_OUT...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Restoration of Retained Data in the Event of a Power Failure:

- If there is no retained data in the Data Memory:
  1. If there is no backed-up data in the Flash-RAM, then use the default data in the Boot Project in the Flash-RAM
  2. If there is backed-up data in the Flash-RAM, then restore this backed-up data to both the Data Memory and the Main Memory.

- If there is retained data in the Data Memory:
  1. If the retained data is consistent with the control application (i.e., not corrupted), then load the data as is from Data Memory to Main Memory.
  2. If the retained data is not consistent with the control application (i.e., it is corrupted), then restore data from the Flash-RAM to both Data Memory and Main Memory.
2.2.3 Multi-tasking

The control software is multi-tasking so that different tasks at different scan rates and priorities will be executed at the scheduled moments.

Tasks are discussed in section 2.3.3, and are an intrinsic part of the multi-tasking system. A task can be:

- **Cyclic** – a periodic event of a fixed scan rate (10 msec minimum)
- **Default** – not periodic but executed when no other tasks are running
- **System** – operating system tasks launched when system events such as cold/warm/hot start, system errors, etc, occur.

**Priority** – If the system is heavily loaded, not all cyclic events may be completed in the given time. The higher a priority a task has, the more time it is given to complete the task. A default task may not get the chance to execute at all if it is too low a priority and the system is heavily loaded.

Task priority is a value between 0 ~ 15. The smaller the number, the higher the priority. The default it 0, which is the highest priority.

The schematic below shows an example of how different tasks are executed during the time cycle of the FCX:

![Task Priority Diagram]

- **Fast Task**: Periodic task with time interval of 10ms, attached program P1
- **Main Task**: Periodic task with time interval of 25ms, attached programs P3, P4
- **Default Task**: Periodic task without fixed time interval, attached programs P5, P6
- **ProConOS**: System Tasks e.g. for communication, debug etc.
- **Task ready**

Task signals (Task will change to state “ready”)
2.2.4 IP Address Setting

FCX Settings are accessed through the Logic Designer by double-clicking on the Target Setting file in the Configuration folder:

Enter either the IP address (as shown in the above example), or the hostname of the FCX, if this exists in the LM Hosts file of the PC. If the IP address of the FCX is not known or needs to be set-up, then follow the procedure in section 2.2.5.1.
Because the FCX can have a dual-redundant ethernet connection, the IP addressing consists of more than one IP address. The IP addresses are known as:

- **VIP** – Virtual IP address
- **PIP** – Physical IP address

It is only necessary to set the VIP address in the FCX. The PIP addresses are set automatically (the PC is another matter as explained next page).

If there is one CPU in the FCX, there are two PIP addresses, one for each ethernet port. These are calculated by incrementing the third IP number as the following example shows:

\[
\text{192.168.x.1}
\]

where:

- \( x = 0 \) (VIP)
- \( x = 1 \) (PIP, port 1)
- \( x = 2 \) (PIP, port 2)

If there are two CPUs, then there are 1 or 2 PIP addresses for the second card as well. In this case, the fourth IP number is incremented by 128 as follows:

\[
\text{192.168.x.129}
\]

- Set by the **APC Command Transfer function** in the Tools menu of Resource Configurator (see section 2.2.5.1).

Note: for the first CPU, do not set the fourth IP number above 128, as this is used for the second CPU. Also, number 127 cannot be used because this would cause the second CPU address to be 255 which is the broadcast address.
Because the FCX has both a Virtual IP address and a Physical IP address (or more), then the IP addressing in the PCs which have the Logic Designer, Resource Configurator and VDS in them must also set multiple addresses, as follows:

1. Single Ethernet, single or dual FCX CPU:

Set two IP addresses for the ethernet port, a Virtual and a Physical address.

Example: 192.168.0.100
          192.168.1.100

2. Dual Ethernet, Single or Dual FCX CPU:

There will be a second ethernet port in the PC. Set up the first port as shown above. For the second port, do not set a Virtual IP address, only Physical IP address.

Example: 192.168.2.100
2.2.5 Resource Configurator

The Resource Configurator is used to set up the FCX online.

The following items are configured:

- CPU – general setup
- CPU – license setting
- IO Module – device label configuration

The Resource Configurator also has the following functions:

- Import/Export existing configuration
- IP address setting
- Upload/Download
- APC – All Program Copy

Components of the Resource Configurator:
2.2.5.1 Procedures - Connecting to the FCX

If the FCX IP address is already setup within the FCX and is known to the user, then the Configurator will connect to the FCX when the HOST is selected in the CONNECT dialog.

• **Username:** stardom
• **Password:** YOKOGAWA

If the FCX IP address is not setup, or the IP address is unknown, then the address needs to be setup as follows:

1. Cycle power to the FCX. The HRDY and RDY lights will flash on the CPU.
2. While they are flashing, press the SHUTDOWN button on the CPU with a pin.
3. The HRDY lamp continues to flash at the same rate, but the R DY lamp will go to a slow flash. This indicates that the FCX is in IP address setting mode.
4. Start Resource Configurator
5. Make sure that “message area” is visible (if not click VIEW – Message Area)
6. After about 30 seconds, a message will appear to say that the FCX has been found.
7. Click on FILE – IP Address Setup and type in the required IP address of the FCX.

Note, the FCX can be identified by the MAC address on the side of the CPU module for FCN, and on the back of the unit for FCJ.

If the FCX has a dual-redundant CPU configuration, then the second CPU IP address has to be set using the APC function in the Tools menu. APC means All Program Copy and copies the contents of the first CPU to the second. This also causes the CPU address to be set.
2.2.5.2 CPU General Setup

The Basic Setup tab provides the following setup options:

SB Bus Redundancy – the SB bus is the bus connecting the nodes of an FCN (maximum 3). Check this box to specify redundant bus connection. Not applicable for FCJ.

Hard Backup of Retained Data – Function Block parameters can be retained when there is no power to the FCX. These are known as Retentive Data and are designated as such when the variable is created (see section 2.4.4). Check this box to ensure that this data is held in SRAM.

Use Java Program – Check this box to enable Java applications to run on the FCX.

Enable I/O Module Auto-Load - If an I/O module is replaced, the setup information of the I/O module which was previously downloaded will be automatically re-loaded. If the user does not want the replaced I/O module to start up immediately after replacement, remove the check mark from this specification. In this case, however, it is necessary to download the setup information of the I/O module from the Resource Configurator.

Network - This function is used to periodically execute the network diagnosis. If the network has a redundant configuration, be sure to specify this function. The diagnosis period should be specified as greater than 500 msec at 100 msec increments.
2.2.5.3  CPU License Setting

When an FCX is first connected to the network, its license needs to be downloaded.

1. Click on the License Tab.
2. Click on “Accessing License File…”
3. Select the files from the floppy disk.
4. Select FILE – DOWNLOAD to download the licenses to the FCX.
5. Restart FCX (Power OFF → ON)
2.2.5.4 I/O Module Configuration

The Resource Configurator automatically identifies the modules installed in the FCX. These are listed in the module layout window, and their details can be viewed by selecting one of the modules.

In an FCN, there are up to 3 I/O racks known as **UNITS**. Each unit contains up to 8 I/O modules, as shown above.

**Device Label** – Each input or output is assigned a Device Label. This must match the Device Label Names configured in the Logic Designer (see section 2.2.1).

2.2.5.5 Import/Export of configuration

Once the I/O Device Tags are setup as required, the information can be exported as an XML file to disk, and downloaded in the event of loss of data in the FCX. The information can also be downloaded to a new FCX with the same hardware specifications.

Import/Export can be found in the FILE menu.

Note that the data cannot be exported in a form that can be used by the Logic Designer. However, it can be copied and pasted into a spreadsheet and formatted there for use.
2.2.5.6 APC - All Program Copy

The APC command forces a copy from one CPU to another in an FCN. Specifically, the command forces a transfer of all program and data from the control CPU to the stand-by CPU. This command is not used for the FCJ.

A CPU can be loaded or put into offline maintenance while the other CPU continues to control the process. At a convenient moment, control can be switched to the new CPU and the APC command can cause it to transfer its program to the other CPU.
2.2.6 The Maintenance Web Page

The maintenance web page in the FCX provides important configuration information about the FCX hardware and software settings. It also provides back-up and restore tools for the FCX programs and settings.

To access the maintenance page of the FCS:

1. Open a web browser and type in the IP address of the FCX, followed by ‘/MNT’
   example: http://172.168.0.1/MNT

2. Username - stardom
   Password - YOKOGAWA

3. A front page appears that provides basic information about the FCX.
4. Click on ‘Maintenance Menu’

The following functions are available:

- System Setting File – display and set system parameters
- System Operation – rebooting and access to back-up tools

Rebooting

It is necessary, when editing system setting files to reboot into Maintenance mode. Remember that this stops the control functions in the FCX. These will resume when rebooting back into online mode.

To reboot:

1. Click on reboot.
2. Select reboot type (e.g. Reboot(Maintenance Mode) to start-up FCX in maintenance)
3. Repeat this procedure to get back to online mode, or to change the IP address (see section 2.2.4)
Back-up tools

The back-up tools are DOS based utilities that enable a complete dump of the FCX files into the PC, and restoration of the files back to the FCX.

Click on “Download the Back-up Tools” to download the tools to the PC. This downloads the self-extracting file “FCXTOOL.EXE”. Move this to the desired directory and double click to extract the back-up and restore tools, FcxBackup.bat and FcxRestore.bat.

To run the Back-up utility, run CMD from the RUN command in the windows start menu, go to the directory where the tools reside and type:

```
fcxbackup <ip_address>
```

e.g. fcxbackup 172.168.0.1

Hostfile Setting

Rather than refer to FCX’s by their IP address, it is possible to refer to them by a name, known as a ‘host name’. This is useful when setting up communications between FCX’s (see section 2.6.8).

The host names are specified in the hosts file of the FCX. In this file, specify the host name and IP address for each FCX on the network. This process must be repeated for each FCX.

The procedure for creating hosts names is as follows:

1. Reboot the FCX into Maintenance Mode (i.e. the editing of the host file cannot be done online).
2. Select EDIT under System Setting File.
3. Select HOSTS File and click OK.
4. Edit the Hosts file page, entering the host name and IP address for each FCX.
5. Click OK and return to the Maintenance Menu.
6. Reboot into online mode.
2.3 Logic Designer Functions

2.3.1 Overview

The Logic Designer is the programming interface for the FCX. It has an Explorer type interface with the top folder being the Project. Underneath that are the folders representing functional sections of the project.

The main folders for the project are:

- **Libraries** – the folder in which all function libraries are registered
- **Data Types** – the folder in which extra data types and structures are registered
- **Logical POUs** – the folder in which all programs and function blocks reside
- **Physical Hardware** – contains the Configurations for the hardware. Each configuration contains FCXs, each with their set of tasks, I/O and global variables.
2.3.2 Program Organization Units

Programs are encapsulated in a Program Organization Unit (POU). A POU can be created from any combination of the five IEC-61131 programming languages, and may be designated as a function, function block, or program.

- **Function Block** - If a POU is designated as a function block it can be used in other POUs, therefore allowing layering of functions. Such a POU can be used as many times as is needed in other POUs. They can also be password protected so that they cannot be edited.

- **Function** – A function is similar to a function block, except that it does not have internal variables (data items). It only has inputs, and a resultant output. An example of a function is an ADD function.

- **Program** - For a POU to execute in the FCX, it must be designated as a program. A program is then assigned to a task in an FCX in the Physical Hardware folder. A function block POU will not execute in its own right. It must first be placed as a function block on a program POU to be executed.

POUs are created and stored in the Logical POU folder, as shown:

Each POU contains three files:

- **Information file** (nameT) – text file with user comments about the function
- **Variables file** (nameV) – all the variables created in the program
- **Program file** (name) – the file with the program in it

(where ‘name’ is the name of the POU)
2.3.3 Tasks

A POU designated as a Program is ‘instantiated’ in the FCX as a task, that is, the POU is assigned to a task. This assigns the following attributes to the program:

- Which FCX it runs in. A program can run in more than one FCX.
- How fast it runs.
- What priority it has.
- The task type.
- How much memory is allocated to it in the stack.

Before a POU can be assigned to a task, a task has to be created. Tasks are created for each FCX in the configuration. Tasks, FCXs and Configurations relate to each other in the following ways:

- **Physical Hardware** – the top folder for the project resources.
- **Configurations** – the next level of folders are Configurations comprising a number of FCXs.
- **Resource** – within a Configuration are a number of FCXs, the actual hardware, known as Resources.
- **Tasks** – within each FCX is the Tasks folder containing a set of tasks.
- **Program Instances** – within each task are a number of Program Instances, i.e., program POUs. When a program POU is assigned to a task, this is termed ‘instantiated’. A program POU can be assigned to more than one task in the FCX, and several programs can be assigned to one task.
A task can be one of three types:

- **Default** – Operates as fast as possible, not at any pre-determined rate.
- **Cyclic** – Operates at a pre-determined rate. Whether it is completed in time depends on its priority. These are deterministic tasks.
- **System** – System tasks are launched due to a system event such as a cold/warm/hot start, bus error, etc.

This relates to the multi-tasking capabilities of the FCX. See section 2.2.3 for more information regarding the multi-tasking functionality of the system.
2.3.4 Operations

This section covers Logic Designer operations to be found in the following menus:

- BUILD – Compiling of programs and Rebuilding of projects.
- ONLINE – Downloading of programs, and online control of the FCX.

This section also covers other related Logic Designer operations, such as:

- Saving Tuning Parameters
- ‘Online’ downloading using ‘Patch POU’
- Project backups

2.3.4.1 The Build Menu

The Build menu provides the following functions:

**Make** – Compile the changed programs in the project and create the necessary source files for downloading to the FCX.

**Patch POU** – An ‘Online’ load function that compiles and downloads the modified program without stopping the FCX. There are restrictions to its use, and these are described in section 2.3.4.3.

**Rebuild Project** – Recompile all of the programs, including declared variables, licenses and all ancillary configuration items. This is necessary after a change has been made to the resources in the project. For example, if the IP address of an FCX is changed, or if an extra license is loaded to it, then the project needs to be rebuilt.

**Build Cross References** – Create a list of all tags and where they are referenced.
2.3.4.2 The Online Menu

The Online Menu provides the following functions:

![Online Menu Screenshot]

**Debug** – Debug sets the FCX into debug mode. The FCX must be running and communicating to the PC for Debug to operate. When running, data can be monitored and set from the program sheets in Logic Designer.

**Project Control** – Controls the operation of the FCX.

- **Stop** – stops the programs running in the FCX
- **Cold** – Performs a cold start of the FCX. All data are initialized.
- **Warm** – Performs a warm start of the FCX. All non-retentive data are initialized.
- **Hot** – Performs a hot start of the FCX. No data is initialized.
- **Reset** – Delete the loaded project in the FCX
- **Download** – Download programs to the FCX
- **Upload** – Upload the zipped project source in the FCX (see 2.3.4.5)
- **Error** – List of errors in the FCX operations
- **Info** – System information about the PLC operations
The **Download** function has the following options:

![Download function interface](image)

There are two main areas to which a project can be downloaded:

- **Project** - The Project is the currently running project in the Main Memory (see section 2.2.2). If a download is carried out to the Project, the FCX stops operation while the download takes place, and restarts with the new program. Note that the restart must be done manually by clicking on cold, warm or hot start.

- **Bootproject** - The Bootproject is the area within the Flash RAM (see section 2.2.2). Downloading to this area does not stop the FCX control functions, but the new program will not become active until power is cycled to the FCX, or the ACTIVATE button is pressed.
The various options are as follows:

- **Download** – download the programs to the selected FCX.
- **Include OPC data** – make sure that all OPC data is registered in the FCX for data access. It is recommended that this is always selected.
- **Download Source** – zip the project and load it as a backup within the FCX. It resides in the Compact Flash. (see 2.3.4.5)
- **Include User-Libraries** – When downloading a Source Project, include all User-Libraries in the zipped project.
- **Include Pagelayouts** – When downloading a Source Project, include all Pagelayouts in the zipped project.
- **Delete Source on Target** – delete the Source Project in the FCX. Only active if there is a Source Project in the FCX.
- **Activate** – Load a Bootproject into Main Memory as the active project. This overwrites the currently running project in the Main Memory, and causes the control functions to stop. These must then be restarted manually.
- **Delete On Target** – Delete the Boot Project currently residing in the Flash RAM.
- **Download File** – download a file to the FCX. This can be any file, including an html file for web browsing.
2.3.4.3 Online Loading with Patch POU

To patch a POU means that changes you have made in a project are compiled, the related code is generated and downloaded automatically to the Controller in one step. During the patch process the Controller keeps on running, i.e. the code execution on the Controller is not aborted while compiling and downloading the changes.

The menu item 'Patch POU' is only available if you switch the corresponding worksheet in offline mode by clicking on the icon 'Debug on/off' in the toolbar.

The 'Patch POU' command can be used for two main purposes:

- **Correcting detected errors** - If you have detected a programming error using the online mode and you have switched to offline mode to remove the programming error you can use 'Patch POU' to compile and load these changes.

- **Further developing a base project** - In some cases it is an advantage to develop a project without stopping the control functions. After 'Making' a base project the first time and downloading it, you can replenish this raw program (i.e. program shell) by editing the various POU code worksheets and patch them to the Controller program which is already running on the target.

In both cases the changes are downloaded automatically while the Controller keeps on running so that you can view them immediately after switching into online mode. Some rules and restrictions have to be observed, and the memory reserve settings must be defined accordingly. See below for more information.

- **How to patch a POU:**

  1. Make sure, that the memory reserve of the POU is large enough to patch the user data (see note below). If it is required to increase the reserve size, the command 'Patch POU' can not be applied. In this case you have to use 'Make' to compile the changed project.

  2. Make sure that the changed worksheet is the active window.

  3. Make sure that the worksheet is in offline mode.

  4. Edit your worksheet or correct any detected programming errors. Observe the rules and restrictions before editing.

  5. Click on the submenu 'Build' and select the menu item 'Patch POU' or click on the corresponding icon in the toolbar.

  6. The compilation process is started and its progress is displayed in the message window.
Memory Reserve Requirements:

When patching, the system stores the user data of the patched POU in the memory reserve defined for these POUs. If there is not enough reserve available, an error message appears and the patch process is aborted.

The default setting is 20% of the available Main Memory for patching. This is a global setting that is available through the Settings option of the resource. In addition, the amount of reserve memory can be set for each POU by right-clicking on the POU in the tree and selecting properties. See the help manual for more information (as described below).

If 'Patch POU' fails:

If the 'Patch POU' command fails, i.e. errors are reported in the message window, because one of the rules/restrictions was not obeyed while patching or there is not enough memory reserve available, you always have to compile the project using 'Make' and initiate a new download. When compiling with 'Make' the data memory in the Controller is assigned newly, i.e. the reserve memory used for earlier patches is released.

When working with several resources, each instance (i.e. resource) has to be patched separately by switching the corresponding online worksheet offline, changing the code and executing the 'Patch POU' command.

⇒ Further Information:

For detailed information on the Patch POU function, go to the help menu in Logic Designer, and the information can be found in the following area:

-  \( \rightarrow \) Contents (runs the Help display)  
-  Editing and developing a project  
  -  Compiling a project  
    -  How to compile a project  
    -  Patch POU
2.3.4.4 Saving Tuning Parameters

In IEC61131 terminology, tuning parameters are known as ‘retained data’, and detailed information on this is given in Section 2.2.2. This section describes the procedures for saving and backing-up tuning parameters.

A tuning parameter is a variable that has its Retain attribute set.

- If this is set, then the data is held in memory.
- It is held in Main Memory (volatile memory) unless the “enable Hard-backup for retained data” is set in the Resource Configurator, in which case it is held in Data Memory.

Using a set of global variables, data can be saved to the Flash Card, and restored from it:

<table>
<thead>
<tr>
<th>Global Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GS_RETAIN_SAVE_SW</td>
<td>Saves data to the Flash RAM card (set to TRUE)</td>
</tr>
<tr>
<td>GS_RETAIN_RESTORE_SW</td>
<td>Restores data from the Flash RAM card (set to TRUE)</td>
</tr>
<tr>
<td>GS_RETAIN_SAVE_PROGRESS</td>
<td>Save progress information (Integer)</td>
</tr>
<tr>
<td>GS_RETAIN_RESTORE_PROGRESS</td>
<td>Restore progress information (Integer)</td>
</tr>
</tbody>
</table>

There is no facility for saving the tuning parameters to the PC.

2.3.4.5 Backing-up projects

Project back-up is available using the project ‘zip’ function and downloading/uploading the zipped file to/from the controller.

- To zip a project, select “Save Project As/Zip Project As” from the FILE menu. Select ‘Save As Type’ = “Zipped Project File, *.wzt”, and a zipped project is created. This can be stored elsewhere as a back-up project.

- To unzip a project, select “Open Project/Unzip Project” from the FILE menu. Select Zipped Project File type. The project is then unzipped into a usable project and can overwrite the currently open project, or appear as a new project.

**Download Source** - A project can be zipped and downloaded to the FCX and stored in the Flash Card through the download dialog (**Download Source**). This effectively ensures makes the back-up project portable, in that anyone can connect to the controller and upload the project for editing.

**Upload** - A zipped project that has been downloaded to the FCX can be uploaded using the **Upload** tool in the Online dialog. The project is then uploaded into the PC and unzipped.

**Maintenance Back-up** – back-up and restore facilities are available through the Maintenance web page as discussed in section 2.2.6. These functions enable a complete dump of the controller memory to PC, and the restoration of this back to the controller.
2.3.5 Procedures

To create a new project:

1. Select FILE - NEW PROJECT
2. The New Project Template selector appears:

3. Select STARDOM PAS. Selecting either of the other two creates a project without the Yokogawa Function Block libraries, which would have to be added manually.
To add a library:

1. Right click on ‘Library’ to insert library, or select ‘Library’ and click on the PROJECT menu and select ADD LIBRARY.

![Image of library selection]

2. Select the required library (.mwt file) and click OK.

![Image of library selection dialog box]
To create a new POU:

1. Right click on ‘Logical POU’, or select it and click on the PROJECT menu.
2. Select one of the three POU types.

3. The following dialog appears:

![Insert dialog](image)

- Name: MOTORDFIVE
- Language: IL
- PLC type: <independent>
- Processor type: <independent>
4. Enter a name for the POU (no spaces), and select the language for the block.

5. If **Function** is selected, note that language type ‘SFC’ cannot be selected. Also, the **Datatype** of the result (return value) has to be specified.

6. It is recommended that the **PLC type** and **Processor type** be selected as <independent> to ensure portability of the configuration.

7. These values can all be changed at any time by right clicking on the POU and selecting **PROPERTIES**.
To Create a Configuration

1. Right click on Physical Hardware and select INSERT - CONFIGURATION

2. Enter the Configuration name (PLC type is always IPC_32).
To Create a Resource

1. Right click on a Configuration and select INSERT - RESOURCE

![Diagram showing right-clicking on Configuration and selecting INSERT - RESOURCE]

2. Enter the Resource name (Processor type is always FCX).

![Insert dialog box showing Resource entry and settings]

These parameters can be changed after the FCX is created by right clicking on the resource and selecting PROPERTIES.

Other parameters for the FCX can be set by right clicking on the resource and selecting SETTINGS.
To Create a Task

1. Right click on a Resource (FCX) and select INSERT - TASK

2. Enter the Task name and Task type.

These parameters can be changed after the task is created by right clicking on the task and selecting PROPERTIES.
3. Enter task priority and scan rate through the next dialog that appears:

![Task settings for IPC_32 dialog box](image)

These parameters can be changed after the task is created by right clicking on the task and selecting SETTINGS.

The Priority is discussed in section 2.2.3, and is a value between 0 ~ 15.
2.4 Data

2.4.1 Data Types

The data types are described in detail in the IEC61131-3 Programming course. A summary is given here:

<table>
<thead>
<tr>
<th>Category</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer Numbers</td>
<td>SINT</td>
<td>Short integer</td>
</tr>
<tr>
<td></td>
<td>INT</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>DINT</td>
<td>Double integer</td>
</tr>
<tr>
<td></td>
<td>USINT</td>
<td>Unsigned short integer</td>
</tr>
<tr>
<td></td>
<td>UINT</td>
<td>Unsigned integer</td>
</tr>
<tr>
<td></td>
<td>UDINT</td>
<td>Unsigned double integer</td>
</tr>
<tr>
<td>Real Numbers</td>
<td>REAL</td>
<td>Real numbers</td>
</tr>
<tr>
<td>Time and Date</td>
<td>TIME</td>
<td>Time duration</td>
</tr>
<tr>
<td>Strings</td>
<td>STRING</td>
<td>Character strings</td>
</tr>
<tr>
<td>Bit Strings</td>
<td>BOOL</td>
<td>Bit string of 1 bit</td>
</tr>
<tr>
<td></td>
<td>BYTE</td>
<td>Bit string of 8 bit</td>
</tr>
<tr>
<td></td>
<td>WORD</td>
<td>Bit string of 16 bit</td>
</tr>
<tr>
<td></td>
<td>DWORD</td>
<td>Bit string of 32 bit</td>
</tr>
<tr>
<td>Boolean</td>
<td>BOOL</td>
<td>switch</td>
</tr>
</tbody>
</table>
2.4.2 Data Structures

Data Structures are a means of encapsulating a series of data of given data types into a single object. Thus, a function block will contain a series of data items that are defined as part of the data structure for the block. For example, the PAS_POU block, PAS_PVI, has the following data structure defined:

```
TYPE (*PVI*)
SD_PPARAM_PVI : STRUCT
  MODE : DWORD;
  ALRM : DWORD;
  AF : DWORD;
  AOF : BOOL;
  PV : CDATA_REAL;
  PVCAL : BOOL;
  RAW : CDATA_REAL;
  SUM : SD_PSUM_DEF;
  HH : REAL;
  LL : REAL;
  PH : REAL;
  PL : REAL;
  VL : REAL;
  PVP : REAL;
  DPV : CDATA_REAL;
  AOFS : DWORD;
END_STRUCT;
END_TYPE
```

This data structure defines all of the parameters within the block. See the following pages for a list of the data structures used for PAS_POU function blocks.

In addition, all process I/O are associated with data structures. These are described in more detail below.
Some of these data items are actually data structures as well. For example, the PV is a data structure called CData_REAL and contains a series of values as follows:

```plaintext
TYPE
  CData_REAL : STRUCT
    Value : REAL;
    Status : DWORD;
    CInfo : DWORD;
    Dummy : DWORD;
    SH : REAL;
    SL : REAL;
    Unit : IndUnit;
END_STRUCT;
END_TYPE
```

Therefore, when referencing the PV in the PVI block, the format is:

```
Tag.PV.Value
```

Note that Unit is also a declared data type, although in this case it is an ARRAY, not a STRUCTURE:

```plaintext
TYPE
  IndUnit : ARRAY[0..7] OF BYTE;
END_TYPE
```

PAS POU blocks also have inputs and outputs that are of particular data types. It is important to know these data types when connecting variables to blocks, or when connecting blocks together.

These can be determined during configuration by opening the function block properties.
List of PAS_POU function blocks and their associated data structure:

<table>
<thead>
<tr>
<th>PAS_POU BLOCK</th>
<th>DATA STRUCTURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAS_ASTM1</td>
<td>SD_PPARAM_ASTM</td>
</tr>
<tr>
<td>PAS_ASTM2</td>
<td>SD_PPARAM_ASTM</td>
</tr>
<tr>
<td>PAS_AS_H/L/M</td>
<td>SD_PPARAM_AS</td>
</tr>
<tr>
<td>PAS_AVE_C</td>
<td>SD_PPARAM_AVE_C</td>
</tr>
<tr>
<td>PAS_AVE_M</td>
<td>SD_PPARAM_AVE_M</td>
</tr>
<tr>
<td>PAS_BDBUF_R</td>
<td>SD_PPARAM_BDBUF_R</td>
</tr>
<tr>
<td>PAS_BDBUF_T</td>
<td>SD_PPARAM_BDBUF_T</td>
</tr>
<tr>
<td>PAS_CT</td>
<td>SD_PPARAM_CT</td>
</tr>
<tr>
<td>PAS_DLAY</td>
<td>SD_PPARAM_DLAY</td>
</tr>
<tr>
<td>PAS_FSUM</td>
<td>SD_PPARAM_FFSUM</td>
</tr>
<tr>
<td>PAS_FSUM_BL</td>
<td>SD_PPARAM_FFSUMBL</td>
</tr>
<tr>
<td>PAS_FOUT</td>
<td>SD_PPARAM_FOUT</td>
</tr>
<tr>
<td>PAS_FUNC_VAR</td>
<td>SD_PPARAM_FUNC_VAR</td>
</tr>
<tr>
<td>PAS_LDLAG</td>
<td>SD_PPARAM_LDLAG</td>
</tr>
<tr>
<td>PAS_MLD</td>
<td>SD_PPARAM_MLD</td>
</tr>
<tr>
<td>PAS_MLD_BT</td>
<td>SD_PPARAM_MLD_BT</td>
</tr>
<tr>
<td>PAS_MLD_PB</td>
<td>SD_PPARAM_MLD_PB</td>
</tr>
<tr>
<td>PAS_ONOFF</td>
<td>SD_PPARAM_ONOFF</td>
</tr>
<tr>
<td>PAS_ONOFF_G</td>
<td>SD_PPARAM_ONOFF_G</td>
</tr>
<tr>
<td>PAS_PG_L30</td>
<td>SD_PPARAM_PG_L30</td>
</tr>
<tr>
<td>PAS_PG_L30BP</td>
<td>SD_PPARAM_PG_L30_BP</td>
</tr>
<tr>
<td>PAS_PID</td>
<td>SD_PPARAM_PID</td>
</tr>
<tr>
<td>PAS_PI_HLD</td>
<td>SD_PPARAM_PI_HLD</td>
</tr>
<tr>
<td>PAS_PVI</td>
<td>SD_PPARAM_PVI</td>
</tr>
<tr>
<td>PAS_P_CLF</td>
<td>SD_PPARAM_PCFL</td>
</tr>
<tr>
<td>PAS_RATIO</td>
<td>SD_PPARAM_RATIO</td>
</tr>
<tr>
<td>PAS_RATIO_RT</td>
<td>SD_PPARAM_RATIO</td>
</tr>
<tr>
<td>PAS_SIO_11/SIO_12/SIO_21/SIO_22/SI_1/SI_2/SO_1/SO_2</td>
<td>SD_PPARAM_SIO</td>
</tr>
<tr>
<td>PAS_SW13/31</td>
<td>SD_PPARAM_SW</td>
</tr>
<tr>
<td>PAS_SW19/91</td>
<td>SD_PPARAM_SW</td>
</tr>
<tr>
<td>PAS_TM</td>
<td>SD_PPARAM_TM</td>
</tr>
<tr>
<td>PAS_TP_CFL</td>
<td>SD_PPARAM_TPCFL</td>
</tr>
<tr>
<td>PAS_T_CFL</td>
<td>SD_PPARAM_TCFL</td>
</tr>
<tr>
<td>PAS_VELLIM</td>
<td>SD_PPARAM_VELLIM</td>
</tr>
<tr>
<td>PAS_VELLIM_PB</td>
<td>SD_PPARAM_VELLIM</td>
</tr>
</tbody>
</table>

The data parameters contained in these data structures can be found in the “SD_PASPOU_PF” library, “Data Types” folder, as described below.
**Process I/O Data Structures:**

In addition to PAS POU function blocks, process I/O also have data structures.

The data structures are as follows:

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTag_I_Anlg</td>
<td>Standard Analog Input with integer value</td>
</tr>
<tr>
<td>DTag_I_Pcnt</td>
<td>Analog input with value in percent</td>
</tr>
<tr>
<td>DTag_I_PulsL</td>
<td>Pulse input (used for analog input from pulse input module)</td>
</tr>
<tr>
<td>DTag_I_PushB</td>
<td>Digital input for Pushbutton Input module with counter value</td>
</tr>
<tr>
<td>DTag_I_Sts</td>
<td>Standard Digital Input</td>
</tr>
<tr>
<td>DTag_I_Temp</td>
<td>Temperature Input (with signal conversion)</td>
</tr>
<tr>
<td>DTag_O_Anlg</td>
<td>Standard Analog Output</td>
</tr>
<tr>
<td>DTag_O_Sts</td>
<td>Standard Digital Output</td>
</tr>
</tbody>
</table>

Details of the data structures can be found in:

`SD_FIELD_PF → DEVICE_TAG_TYPE`

The four ‘standard’ process I/O data structures are described here:

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Element</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTag_I_Anlg</td>
<td>Attr</td>
<td>WORD</td>
<td>Data Attribute</td>
</tr>
<tr>
<td>DTag_I_Anlg</td>
<td>Value</td>
<td>UINT</td>
<td>Binary data 0x1000 - 0x5000</td>
</tr>
<tr>
<td></td>
<td>Status</td>
<td>WORD</td>
<td>Data Status (Good, Bad, etc)</td>
</tr>
<tr>
<td>DTag_O_Anlg</td>
<td>SH</td>
<td>REAL</td>
<td>Scale High</td>
</tr>
<tr>
<td></td>
<td>SL</td>
<td>REAL</td>
<td>Scale Low</td>
</tr>
<tr>
<td></td>
<td>Unit</td>
<td>IndUnit</td>
<td>Engineering Unit</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Element</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTag_I_Sts</td>
<td>Attr</td>
<td>WORD</td>
<td>Data Attribute</td>
</tr>
<tr>
<td>DTag_O_Sts</td>
<td>Value</td>
<td>WORD</td>
<td>On/Off data</td>
</tr>
<tr>
<td></td>
<td>Status</td>
<td>WORD</td>
<td>Data Status (Good, Bad, etc)</td>
</tr>
</tbody>
</table>

Note that the Digital I/O data Value is a WORD, and therefore must be converted to a BOOL before it can be used in logic functions. However, whenever a digital I/O value is created, a corresponding boolean is also created.

For example, if a digital input is created with name DI1, then another variable with name DI1_BOOL is also created automatically, as a BOOL data type. This can then be used in logic control.
How to determine the data type of a variable:

The data structure of each PASPOU block (Yokogawa Function Block) can be reviewed by:

- clicking on the LIBRARY tab,
- going to the SD_PASPOU_PF folder,
- then double clicking on the relevant function block under Data Types.

Other related data structures for Yokogawa blocks can be found in the SD_FIELD_PF folder. The PAS_POU_TYPE file has important data structures such as $\texttt{CData\_Real}$. 
How to determine the data type of input/outputs of a Function Block:

1. Right-click on the Function Block (or Function) and select Object Properties.
2. Click on the FB/FU tab. The data types of each of the inputs/outputs are displayed.
2.4.3 Variable Types

Data is in the form of **global** variables or **local** variables:

- **Local Variables** – internal data that is used within a POU (function block or program) and not required to be accessed elsewhere. This can be in the form of a Variable, or Input/Output Variable.

- **Global Variables** – data that is available to all POUs in the FCX. It is declared as an **External Variable** within a POU, and as a **Global Variable** in the Resource (FCX) folder.

Data within function blocks are assigned as follows:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAR</td>
<td>Local Variable</td>
<td>Internal Data</td>
</tr>
<tr>
<td>VAR_EXTERNAL</td>
<td>Global Variable</td>
<td>Data to be used in other blocks</td>
</tr>
<tr>
<td>VAR_IN_OUT</td>
<td>An Input/Output Variable</td>
<td>Data that is an input to and output from the function block.</td>
</tr>
<tr>
<td>VAR_INPUT</td>
<td>An Input Variable</td>
<td>Data that is an input to the function block.</td>
</tr>
<tr>
<td>VAR_OUTPUT</td>
<td>An Output Variable</td>
<td>Data that is an output from the function block.</td>
</tr>
</tbody>
</table>

If a variable is used only within a POU it is called **local variable**. In those cases the variable keywords VAR, VAR_INPUT and VAR_OUTPUT are used.

If a variable is be used within the whole project it is called a **global variable**. It has to be declared as VAR_GLOBAL in the global declaration and as VAREXTERNAL in each POU where it is used.
The relationship between local and global variables, and the FCX software configuration is as follows:
### 2.4.4 Declaring Variables

Data must be declared to be used by the system. The Logic Designer declares most data automatically as it is created. This is described in detail in the IEC61131-3 Programming course.

Apart from the standard declaration items, the following specific settings can be defined:

#### Other Data Setup Items:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Usage</th>
<th>Description</th>
<th>Address</th>
<th>Init</th>
<th>Retain</th>
<th>PDD</th>
<th>OPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIC100</td>
<td>PAS_PID</td>
<td>VAR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIC100</td>
<td>PAS_PID</td>
<td>VAR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DUMMY1</td>
<td>STRING_PRM</td>
<td>VAR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEVELIN</td>
<td>CData_REAL</td>
<td>VAR_INPUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLOWIN</td>
<td>CData_REAL</td>
<td>VAR_INPUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PAV</td>
<td>CData_REAL</td>
<td>VAR_OUTPUT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Retain** – this check box selects whether this data is to be retained in SRAM (see 2.2.5.2).

**OPC** – this check box selects whether this data is to be available to the VDS for web graphics. If the data is not selected as OPC it will not be detected when imported through the Object Builder (see section 3.2.2).

**PDD** – This must be set on if the variable is to be communicated to another FCX (see section 2.6.8).
How to Register Data:

1. Double click on a connection point on a function block or logic element of a ladder.
2. Type in the name of the variable, and select either Local and Global. See next page for more information on selecting a Global Variable.
3. In the COMMON tab, select Variable Type (Usage) and Data Type. If the data is to be read by another system, such as the VDS for graphics, then select OPC.
How to Register Global Variable Data:

1. A Global Variable must first be declared in the Global Variable Definition file in the FCX as follows:

   ![Global Variable Definition](image1)

   - **User 1**
     - PSTART
     - PIRUN
     - P2RUN
     - P3RUN
     - P1DUTY

2. This Global Variable can now be used in the POUs.
3. In a POU, double click on a connection point on a function block or logic element of a ladder.
4. Click on the Global Scope tab and select the Resource (FCX) and the category of Global Variable in which the variable is to be placed, in this case, User1.
5. Click on the Contact or Variables tab and type in the name of the variable, and select Global.
6. Everything is grayed out in Common, and OPC cannot be set. This is set in the Global Variables page in the FCX folder.

![Contact/Call Properties](image2)

![Variable Properties](image3)
### 2.4.5 Global Constants

Various constants that are used in the PAS POU blocks and elsewhere are defined in the Global Variables file. These may be used in POUs as required. For example, to write an ‘HH’ alarm to the VDS (see section 2.6.7), or to set the mode of a controller (see section 2.6.4).

*The constants are divided into three categories:*

- **Status Definition** – Definitions that are often used, such as mode and alarm status.

- **Engineering Constant** – Constants used for engineering parameter setup.

- **Application Flags** – Internal constants and connections in PAS POU blocks. Not normally used by the user.

The constants all begin with ‘GM_’. Examples include:

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>GM_MODE_MAN</td>
<td>MAN</td>
</tr>
<tr>
<td>GM_MODE_AUT</td>
<td>AUT</td>
</tr>
<tr>
<td>GM_MODE_CAS</td>
<td>CAS</td>
</tr>
<tr>
<td>GM_ALRM_HH</td>
<td>HH</td>
</tr>
<tr>
<td>GM_ALRM_HI</td>
<td>HI</td>
</tr>
<tr>
<td>GM_ALRM_LO</td>
<td>LO</td>
</tr>
<tr>
<td>GM_ALRM_LL</td>
<td>LL</td>
</tr>
</tbody>
</table>
2.5 IEC61131-3 Programming

2.5.1 Overview

IEC61131-3 specifies five programming languages to be used in the system:

- Function Block Diagrams (FBD)
- Structured Text (ST)
- Ladder Diagrams (LD)
- Sequential Function Charts (SFC)
- Instruction List (IL)

It is beyond the scope of this course to describe these languages, however, Yokogawa has added functions to the system to enhance the control functionality, and these are described here.

Yokogawa has defined a set of function blocks which can be used with FBDs, STs and LDs. These blocks are similar to those used in the CS3000 DCS and enable complex control functions without having to use the basic IEC61131 function blocks.

The libraries of functions and function blocks available are:

<table>
<thead>
<tr>
<th>Libraries</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functions</td>
<td>IEC61131 Functions</td>
</tr>
<tr>
<td>Function Blocks</td>
<td>IEC61131 Function Blocks</td>
</tr>
<tr>
<td>String FUs</td>
<td>String processing functions</td>
</tr>
<tr>
<td>Type Conv. FUs</td>
<td>Conversion between data types</td>
</tr>
<tr>
<td>Bit UTIL</td>
<td>Bit processing in Words</td>
</tr>
<tr>
<td>PROCONOS</td>
<td>Utilities for accessing O/S memory locations</td>
</tr>
<tr>
<td>SD_CF CX_PF</td>
<td>Inter FCX communication</td>
</tr>
<tr>
<td>SD_FC XCOM_LIB</td>
<td>Inter FCX communication</td>
</tr>
<tr>
<td>SD_FC XPAS_LIB</td>
<td>Read/Write data from one function block to another</td>
</tr>
<tr>
<td>SD_FC XSTD_LIB</td>
<td>Yokogawa Utilities</td>
</tr>
<tr>
<td>SD_FC XSYS_LIB</td>
<td>Yokogawa Utilities</td>
</tr>
<tr>
<td>SD_FIELD_PF</td>
<td>Yokogawa Utilities</td>
</tr>
<tr>
<td>SD_PASPARTS_PF</td>
<td>Utilities for the PASPOU function blocks</td>
</tr>
<tr>
<td>SD_PASPOU_PF</td>
<td>Yokogawa Function Blocks</td>
</tr>
<tr>
<td>SD_PASUTIL_PF</td>
<td>Data conversion functions</td>
</tr>
<tr>
<td>SD_WIRE_PF</td>
<td>WDA Utilities</td>
</tr>
<tr>
<td>SD_CFAM3E_PF</td>
<td>Communication Utilities to FA-M3 PLCs</td>
</tr>
<tr>
<td>SD_CMELSECE_PF</td>
<td>Communication Utilities to Melsec PLCs</td>
</tr>
</tbody>
</table>

Note: All ‘SD’ libraries are Yokogawa function libraries
2.5.2 Execution Order

The order of execution for programs is from top left to bottom right. For Structured Text, Ladder Diagrams and Instruction List, the execution order is unambiguous. For SFCs the execution order is defined by the branching structure.

However, for FBDs the execution order can be quite ambiguous, and depends not only on the position of the function/function block on the worksheet, but also how it is connected.

For example, if an analog input function block is placed to the right of the PID block it connects in to, the compiler recognizes the fact that it is an input to the PID block and executes it first.

In the case where there is a loop-back from one block to another, so that there is a continuous loop, the compiler does not know which one to execute first and makes a guess. It is highly recommended that these situations are avoided. Use local variables instead (see section 2.6.4).

It is possible to view the execution order by clicking on LAYOUT → EXECUTION ORDER.
2.5.3 Procedures For Creating Function Block Diagrams

- Creating Function Block Diagrams

1. Create a Logical POU (see section 2.3.4) designated as a function block.
2. Double click on the program file of the POU as follows:

3. The program sheet appears. Onto this the blocks are placed.
4. Open the Edit Wizard window by clicking on VIEW ➔ EDIT WIZARD.
5. The Edit Wizard contains all the library function, including PASPOU blocks.

6. Click on the sheet, and the Edit Wizard will become available. Double click on the required function block.
7. The function block properties appears, and the tagname for the block is entered.
8. To join function blocks together, select the wiring tool, click on the source terminal, and click on the destination terminal:
Wireless connections within drawings:

Where it is inconvenient to connect variables using the standard wiring, the connector function allows data to be connected between function blocks within the same worksheet.

1. Click on the output terminal
2. Click on the Connector button
3. Type in a name, and click OK.
4. Follow same procedure for destination terminal

⇒ See also Section 2.6.4 for other ways of connecting between functions and function blocks.
2.5.4 POU Security

If the worksheet (or POU) is designated as a Function or a Function Block, it can be password protected so that it can be used as a block but not opened or edited. A POU designated as a program can also be password protected, so that the entire program that is running in the FCX is locked out and cannot be accessed by the user without a password.

- **How to set up security:**
  - Right click on the POU folder
  - Select PROPERTIES
  - Select the SECURITY tab
  - Check Write and/or Read protection, and click OK
  - In the FILE menu, select ‘Enter Password’ and enter a new password
  - Close and reopen Logic Designer. The protected POU will have a key next to it and access to the worksheet and data will be restricted.

- **How to modify security configuration:**
  - To edit the POU, select ‘Enter Password’ in the FILE menu and enter the password.
  - The security can be removed by clicking on Remove Password in the Enter Password dialog, or by unselecting Write/Read protection in the Properties.
2.6 Yokogawa PASPOU Function Blocks

2.6.1 Summary of PASPOU Blocks

The PASPOU library consists of function blocks developed by Yokogawa to enable programming of control applications. See Appendix 1 for more details on these blocks.

- Input Output blocks
- Controller blocks
- Manual loader blocks
- Signal setter blocks
- Signal limiter blocks
- Signal selector blocks
- Signal distributor blocks
- Calculation Blocks
- Sequence Auxiliary Blocks

### Input Output Blocks

<table>
<thead>
<tr>
<th>Block</th>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analog Input</td>
<td>PAS_AI_ANLG</td>
<td>Analog input block – Standard type</td>
</tr>
<tr>
<td>Analog Input</td>
<td>PAS_AI_PCNT</td>
<td>Analog input block – Percent input</td>
</tr>
<tr>
<td>Analog Input</td>
<td>PAS_AI_PULS_CI</td>
<td>Analog input block – Pulse input</td>
</tr>
<tr>
<td>Analog Input</td>
<td>PAS_AI_PULS_QT</td>
<td>Analog input block – Temperature input</td>
</tr>
<tr>
<td>Analog Output</td>
<td>PAS_AO_ANLG</td>
<td>Analog output block</td>
</tr>
<tr>
<td>Digital Input</td>
<td>PAS_DI_PUSHB</td>
<td>Digital input – pushbutton type</td>
</tr>
<tr>
<td>Digital Input</td>
<td>PAS_DI_STS</td>
<td>Digital input – status type</td>
</tr>
<tr>
<td>Digital Output</td>
<td>PAS_DO_STS</td>
<td>Digital output – status type</td>
</tr>
<tr>
<td>Digital Output</td>
<td>PAS_DO_STS_PW</td>
<td>Digital output – pulse width type</td>
</tr>
</tbody>
</table>

### Controller Blocks

<table>
<thead>
<tr>
<th>Block</th>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controllers</td>
<td>PAS_PID</td>
<td>PID controller block</td>
</tr>
<tr>
<td>Controllers</td>
<td>PAS_PI-HLD</td>
<td>Sampling PI controller block</td>
</tr>
<tr>
<td>Controllers</td>
<td>PAS_ONOFF</td>
<td>2-position ON/OFF controller block</td>
</tr>
<tr>
<td>Controllers</td>
<td>PAS_ONOFF_G</td>
<td>3-position ON/OFF controller block</td>
</tr>
</tbody>
</table>
### Manual Loader Blocks

<table>
<thead>
<tr>
<th>Block Type</th>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual Loader</td>
<td>PAS_MLD</td>
<td>Manual loader block</td>
</tr>
<tr>
<td></td>
<td>PAS_MLD_BT</td>
<td>Manual loader block with input indicator</td>
</tr>
<tr>
<td></td>
<td>PAS_MLD_PB</td>
<td>Manual loader block with Auto/Man switch</td>
</tr>
</tbody>
</table>

### Signal Setter Blocks

<table>
<thead>
<tr>
<th>Block Type</th>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal Setter</td>
<td>PAS_RATIO</td>
<td>Ratio set block</td>
</tr>
<tr>
<td></td>
<td>PAS_RATIO_RT</td>
<td>30-zone program set block</td>
</tr>
<tr>
<td></td>
<td>PAS_PG_L30</td>
<td>30-zone program set block</td>
</tr>
<tr>
<td></td>
<td>PAS_PG_L30_BP</td>
<td>30-zone program set block</td>
</tr>
</tbody>
</table>

### Signal Limiter Block

<table>
<thead>
<tr>
<th>Block Type</th>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal Limiters</td>
<td>PAS_VELLIM</td>
<td>Velocity limiter block</td>
</tr>
<tr>
<td></td>
<td>PAS_VELLIM_PB</td>
<td>Velocity limiter block</td>
</tr>
</tbody>
</table>

### Signal Selector Blocks

<table>
<thead>
<tr>
<th>Block Type</th>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal Selectors</td>
<td>PAS_AS_H</td>
<td>High auto-selector block</td>
</tr>
<tr>
<td></td>
<td>PAS_AS_M</td>
<td>Medium auto-selector block</td>
</tr>
<tr>
<td></td>
<td>PAS_AS_L</td>
<td>Low auto-selector block</td>
</tr>
</tbody>
</table>

### Signal Distributor Blocks

<table>
<thead>
<tr>
<th>Block Type</th>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal Distributors</td>
<td>PAS_FOUT</td>
<td>Cascade control signal distribution block</td>
</tr>
<tr>
<td></td>
<td>PAS_FFSUM</td>
<td>Feedforward control signal addition block</td>
</tr>
<tr>
<td></td>
<td>PAS_FFSUM_BL</td>
<td>Feedforward control signal addition block</td>
</tr>
</tbody>
</table>
## Calculation Blocks

<table>
<thead>
<tr>
<th>Block Type</th>
<th>Code</th>
<th>Block Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analog Calculation</td>
<td>PAS_LDLAG</td>
<td>Lead/Lag</td>
</tr>
<tr>
<td></td>
<td>PAS_DLAY</td>
<td>Dead Time</td>
</tr>
<tr>
<td></td>
<td>PAS_AVE_M</td>
<td>Moving Average</td>
</tr>
<tr>
<td></td>
<td>PAS_AVE_C</td>
<td>Cumulative Average</td>
</tr>
<tr>
<td></td>
<td>FUNC-VAR</td>
<td>Variable line-segment function</td>
</tr>
<tr>
<td></td>
<td>PAS_P_CFL</td>
<td>Pressure Compensation of gas flow</td>
</tr>
<tr>
<td></td>
<td>PAS_T_CFL</td>
<td>Temperature Compensation of gas flow</td>
</tr>
<tr>
<td></td>
<td>PAS_TP_CFL</td>
<td>Pressure &amp; Temp Compensation of gas flow</td>
</tr>
<tr>
<td></td>
<td>PAS_ASTM1</td>
<td>Oil Temp Correction: Old JIS</td>
</tr>
<tr>
<td></td>
<td>PAS_ASM2</td>
<td>Oil Temp Correction: New JIS</td>
</tr>
<tr>
<td></td>
<td>PAS_SW-13</td>
<td>3-pole 1-position Selector Switch</td>
</tr>
<tr>
<td></td>
<td>PAS_SW-31</td>
<td>1-pole 3-position Selector Switch</td>
</tr>
<tr>
<td></td>
<td>PAS_SW-19</td>
<td>9-pole 1-position Selector Switch</td>
</tr>
<tr>
<td></td>
<td>PAS_SW-91</td>
<td>1-pole 9-position Selector Switch</td>
</tr>
<tr>
<td></td>
<td>PAS_BD_BUF_R</td>
<td>Batch data set block</td>
</tr>
<tr>
<td></td>
<td>PAS_BD_BUF_T</td>
<td>Batch data set block</td>
</tr>
</tbody>
</table>

## Sequence Auxiliary Blocks

<table>
<thead>
<tr>
<th>Block Type</th>
<th>Code</th>
<th>Block Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timers</td>
<td>PAS_TM</td>
<td>Timer</td>
</tr>
<tr>
<td></td>
<td>PAS_CT</td>
<td>Counter</td>
</tr>
<tr>
<td>Switch Instrument Blocks</td>
<td>SI-1</td>
<td>Switch Instrument Block – 1 input</td>
</tr>
<tr>
<td></td>
<td>SI-2</td>
<td>Switch Instrument Block – 2 inputs</td>
</tr>
<tr>
<td></td>
<td>SO-1</td>
<td>Switch Instrument Block – 1 output</td>
</tr>
<tr>
<td></td>
<td>SO-2</td>
<td>Switch Instrument Block – 2 outputs</td>
</tr>
<tr>
<td></td>
<td>SIO-11</td>
<td>Switch Instrument Block – 1 input/1 output</td>
</tr>
<tr>
<td></td>
<td>SIO-21</td>
<td>Switch Instrument Block – 2 input/1 output</td>
</tr>
<tr>
<td></td>
<td>SIO-12</td>
<td>Switch Instrument Block – 1 input/2 output</td>
</tr>
<tr>
<td></td>
<td>SIO-22</td>
<td>Switch Instrument Block – 2 input/2 output</td>
</tr>
</tbody>
</table>
2.6.2 Using PASPOU Blocks

Function block details are covered in Appendix 1. This section covers some important points in using the blocks.

Most of the PASPOU blocks have the following terminals

<table>
<thead>
<tr>
<th>Terminal Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RB_IN</td>
<td>These terminals are connected together between blocks to enable pushback related to cascade connections. In most blocks, RB_OUT contains the CASV (See 2.6.6).</td>
</tr>
<tr>
<td>RB_OUT</td>
<td></td>
</tr>
<tr>
<td>ENG_PRM</td>
<td>Used for setting engineering parameters such as Function Block comment, PID reverse action switch, Engineering Units and range. (See 2.6.4)</td>
</tr>
<tr>
<td>PRM_WT</td>
<td>Used for setting dynamic data (tuning parameters) (See 2.6.4).</td>
</tr>
<tr>
<td>PRM_RD_OUT</td>
<td>Makes an internal data parameter available at this terminal to be read by other function blocks. (See 2.6.4)</td>
</tr>
<tr>
<td>IN/OUT</td>
<td>These terminals connect the normal data flow between blocks. (See 2.6.3)</td>
</tr>
<tr>
<td>CAS_IN</td>
<td>In cascade loops, the OUT of the preceding loop connects to the CAS_IN terminal of the following loop (See 2.6.6)</td>
</tr>
</tbody>
</table>

These are discussed in more detail in the following sections.

In addition, there are various function libraries that provide support functions for the PASPOU blocks. Some of these are also described in the following sections and are summarised as follows:

<table>
<thead>
<tr>
<th>Function Library</th>
<th>Description</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD_FCXPAS_LIB</td>
<td>Read/Write parameters to PASPOU blocks.</td>
<td>2.6.4</td>
</tr>
<tr>
<td>SD_PASUTIL_PF</td>
<td>Converting data types for PASPOU blocks.</td>
<td>2.6.5</td>
</tr>
<tr>
<td>SD_FIELD_PF</td>
<td>Alarm reporting using the PAS_MSG_UPRCALM function.</td>
<td>2.6.7</td>
</tr>
<tr>
<td>SD_FCXCOM_LIB</td>
<td>Inter-FCX Communication</td>
<td>2.6.8</td>
</tr>
</tbody>
</table>
2.6.3 Connecting Process I/O

Analog and I/O are processed through the Analog I/O blocks that are part of the PAS_POU Function block library. The types of blocks available are listed above in section 2.6.1.

- The Analog input blocks (PAS_AI_*) convert the input into an engineering value (a regularized value), and into a ‘CData’ type, as appropriate.

- The Analog output block (PAS_AO_ANLG) converts the 0-100% ‘CData_Real’ type input into a REAL type output value suitable for the analog output module.

An example of how these I/O modules are used is given below:

![Diagram of process input and output with labeled connections](image)
How to apply a Device Label Name:

To apply the Device Label name (analog input or output tag) to the terminal, use the following procedure:

1. Double click on the IN terminal of the Analog Input block.
2. Click on the GLOBAL SCOPE tab.
3. Select DeviceLabel_Input_A (for analog input) or DeviceLabel_Output_A (for analog output).
4. Click on the VARIABLES tab.
5. Click on Global
6. Select the required analog device name and click OK.
2.6.4 Connecting Internal Parameters

Input/Output Connections:

Normal data flow is through the IN and OUT terminals of each function block as the example shows in 2.5.3.1.

Connecting across drawings:

Sometimes it is required to connect normal signal flow across drawings, or even within the same worksheet, but where it is not possible to draw a wiring connection. One method is discussed in section 2.5.2, “Wireless Connection Within Drawings”. In this section we look at a method using a variable. The type of variable to be used depends on the nature of the connection:

- When connecting across POUs the variable must be global.
- When connecting within the same POUs, the variable can be local.

Note that if the local variable is defined in the Program POU scope, then it can be used to transfer data across worksheets within the same program.
Access Parameter Connections:

As well as the normal signal flow, it is often necessary to write to, or read from, a data value within a function block from another function block. There are several ways of achieving this:

- Using the Data Reference (PRM_RD_OUT) and Setup (PRM_WT) terminals on the function block
- Using Read/Write POUs

**Using the Data Reference and Setup terminals:**

![Diagram](image)

PRM_WT – this terminal allows for the writing of a value from another function or function block into the current block. The variable data type is a STRING_PRM.

NOTE: Even if it is not used, a variable must be connected to each PRM_WT terminal.
PRM_RD_OUT – this terminal allows for the setting of data into a variable that can be read by another function block. The variable can be of any name, but it is a structured variable (eg, SD_PPARAM_PID), with a type depending on the function block used. See section 2.4.2 for a list of data types for each PASPOU block.

Because it is a structured variable, it actually contains several variables, and the correct variable must be included in the reference terminal of the reading block.

For example, if the name of the variable in the source block is PRM_RD_PID, then to read the PV from this block, the terminal connection is PRM_RD_PID.PV.

Remember that PV is a data structure itself, so if the destination block takes a real value, then the terminal connection will be PRM_RD_PID.PV.Value.
Using the Read/Write POUs (Library: SD_FCXPAS_LIB):

There are a range of POUs that allow the setting and reading of data between function blocks. These can be found in the **SD_FCXPAS_LIB** library. These are required if the setting or reading is to be controlled by an event. That is, they have a switch input, that when TRUE will allow the read/write transaction to take place.

The structure of a Write block is as follows:

```
SD_FCXP_PRM_WT_REAL

SW01  --- EN --- PARAM_STR

'MV'  --- DATA --- WT_STR

PID_PRM
```

where:

- **EN**: Enable switch (Boolean – TRUE/FALSE)
- **ENO**: Retransmitted value of Enable Switch
- **PARAM_STR**: The data variable to be written to
- **DATA**: The value to be loaded into the data variable
- **WT_STR**: The data structure name connected to the PRM_WT terminal on the destination block.

When the variable connected to **EN** = TRUE, then the value connected to the **DATA** terminal is loaded into the data item connected to the **PARAM_STR**, which is loaded into the data structure variable connected to the **WT_STR**.

In the above example, when **SW01** = TRUE, 50.0 is loaded into the variable **PID_PRM**, which is then loaded into the PID block ‘FIC2000’.

The available write blocks are as follows:

<table>
<thead>
<tr>
<th><strong>SD_FCXP_PRM_WT_BOOL</strong></th>
<th>Boolean (TRUE/FALSE)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SD_FCXP_PRM_WT_DINT</strong></td>
<td>Double Integer</td>
</tr>
<tr>
<td><strong>SD_FCXP_PRM_WT_DWORD</strong></td>
<td>Bit String</td>
</tr>
<tr>
<td><strong>SD_FCXP_PRM_WT_INT</strong></td>
<td>Integer</td>
</tr>
<tr>
<td><strong>SD_FCXP_PRM_WT_REAL</strong></td>
<td>Real Number</td>
</tr>
<tr>
<td><strong>SD_FCXP_PRM_WT_STRING</strong></td>
<td>String Type</td>
</tr>
<tr>
<td><strong>SD_FCXP_PRM_WT_TIME</strong></td>
<td>Time</td>
</tr>
<tr>
<td><strong>SD_FCXP_PRM_WT_UDINT</strong></td>
<td>Unsigned Double Integer</td>
</tr>
<tr>
<td><strong>SD_FCXP_PRM_WT_UINT</strong></td>
<td>Unsigned Integer</td>
</tr>
</tbody>
</table>
Setting the MODE of a controller:

Use the function: **SD_FCXP_PRM_WT_DWORD**

The method is the same as that described above. However, in this case, the DATA is a DWORD.

For example, to set the controller to MANUAL, the DATA value is: 16#00800000.

However, it is possible to use Status Definition constants as defined in the Global Variables file (see section 2.4.5). In this way, self-explanatory names can be used instead of the DWORD variables.

In this example, the data connection is linked to the global variable “GM_MODE_AUT” which contains the value “AUT” as a DWORD.
The structure of a Read block is as follows:

where:

- **EN**: Enable switch (Boolean – TRUE/FALSE)
- **ENO**: Retransmitted value of Enable Switch
- **Result (RST1)**: Result of the read function
- **PARAM_STR**: The data variable to be read
- **WT_STR**: The data structure name connected to the PRM_WT terminal on the source block.

When the variable connected to **EN** = TRUE, then the data structure connected to the **WT_STR** reads the data variable specified by the **PARAM_STR** from the source function block. The result is

In the above example, when SW01 = TRUE, then the PV from FI2000 is read. Note that PV is a CData_Real and so the result (RST1) must be specified as such.

The available read blocks are as follows:

<table>
<thead>
<tr>
<th>Block Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD_FCXP_PRM_RD_BOOL</td>
<td>Boolean (TRUE/FALSE)</td>
</tr>
<tr>
<td>SD_FCXP_PRM_RD_DINT</td>
<td>Double Integer</td>
</tr>
<tr>
<td>SD_FCXP_PRM_RD_DWORD</td>
<td>Bit String</td>
</tr>
<tr>
<td>SD_FCXP_PRM_RD_INT</td>
<td>Integer</td>
</tr>
<tr>
<td>SD_FCXP_PRM_RD_REAL</td>
<td>Real Number</td>
</tr>
<tr>
<td>SD_FCXP_PRM_RD_STRING</td>
<td>String Type</td>
</tr>
<tr>
<td>SD_FCXP_PRM_RD_TIME</td>
<td>Time</td>
</tr>
<tr>
<td>SD_FCXP_PRM_RD_UDINT</td>
<td>Unsigned Double Integer</td>
</tr>
<tr>
<td>SD_FCXP_PRM_RD_UINT</td>
<td>Unsigned Integer</td>
</tr>
</tbody>
</table>
Using the Write POUs (SD_FCXP_PRM_WT_) to set Engineering Parameters in PAS POU blocks:

Setting Engineering parameters such as comment, engineering units and range, reverse/direct action switch in a PID, etc, can only be done through the ENG_PRM.

A Structured Text Function Block is created with a list of all the parameters associated with the function block (such as PID), using the SD_FCXP_PRM_WT_* function. Templates for these already exist in the PAS_EXAMPLE_E project that is supplied with Logic Designer. See also the example in section 2.6.3.

Setting bulk parameters to the block using the SD_FCXP_PRM_WT_* functions:

```
IF (FLAG=TRUE) THEN
  RETURN;
END_IF;

(* Engineering parameter     Set value      Initial value*)
nRet:=SD_FCXP_PRM_WT_BOOL( '_CTL_DIR_SW',     DIR_SW,  ENGPRM);
nRet:=SD_FCXP_PRM_WT_BOOL( '_PVTRK_SW_MAN',   FALSE,   ENGPRM);
nRet:=SD_FCXP_PRM_WT_BOOL( '_PVTRK_SW_AUTOP', FALSE,   ENGPRM);
nRet:=SD_FCXP_PRM_WT_REAL( '_HYS_ALRM_HILO',  2.0,     ENGPRM);
nRet:=SD_FCXP_PRM_WT_REAL( '_HYS_ALRM_HHLL',  HHLL,     ENGPRM);
nRet:=SD_FCXP_PRM_WT_INT(  '_SMPL_NUM',       INT#1,   ENGPRM);
nRet:=SD_FCXP_PRM_WT_TIME( '_SMPL_INTVL',     t#0s,    ENGPRM);
  ..
nRet:=SD_FCXP_PRM_WT_INT(  '_SUM_SW',         SUM_SW,  ENGPRM);
nRet:=SD_FCXP_PRM_WT_REAL( '_SUM_LOW_CUT',    0.0,     ENGPRM);
nRet:=SD_FCXP_PRM_WT_INT(  '_SUM_PV_DP',      INT#0,   ENGPRM);
nRet:=SD_FCXP_PRM_WT_STRING( '_SUM_UNIT',      '',      ENGPRM);
nRet:=SD_FCXP_PRM_WT_STRING( 'COMMENT',       CMT,     ENGPRM);
  ..

FLAG := TRUE;
```

Note that the Initial Values are a mixture of literal values and Variables. The ones designated as variables are circled in the above example. In the variable page of the worksheet, these are declared as VAR_INPUT, and appear as inputs to the function block. Any parameter in this program can be set as a variable and declared as a VAR_INPUT.
2.6.5 Converting Between Data Types

Each variable in the system is assigned a data type or data structure. When these parameters are connected between blocks, they must be of the same data type. There are several data conversion blocks that provide conversion between different data types.

*Data conversion is necessary under the following conditions:*

- Connecting standard IEC Function Blocks and Functions to PAS_POU blocks
- Converting Boolean switches to Integer numbers
- Connecting blocks of different data types

### 2.6.5.1 Connecting between IEC Blocks and PAS_POU blocks

Most IEC blocks have simple data types as variables in their terminal connections. PAS_POU blocks have data structures in the terminal connections (such as CData_Real). Therefore, to connect an IEC block and a PASPOU block together, a data conversion block is required between them.

![Diagram](image)

In the above example, the OUT terminal in the PAS_PVI block contains the PV. The data type of the PV is CData_Real, which is a data structure containing the Value, Status, Range, and Engineering Unit. The PAS_CDR_TO_PV block extracts these individual items and makes them available as simple data types. Therefore, the OUT terminal of this block contains the Value of the PV as a REAL data type. This can then be connected directly to the input of the ADD block as a simple REAL Value.
The Data Conversion blocks are in the **SD_PASUTIL_PF** library and are:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAS_CDR_TO_PV</td>
<td>Convert CData_Real to data (in engineering units)</td>
</tr>
<tr>
<td>PAS_PV_TO_CDR</td>
<td>Convert data to CData_Real (non-engineering units)</td>
</tr>
<tr>
<td>PAS_CDR_STR</td>
<td>Convert data to CData_Real (Analog Structuralization)</td>
</tr>
<tr>
<td>PAS_CDR_DESTR</td>
<td>Convert CData_Real to data (Analog De-structuralization)</td>
</tr>
<tr>
<td>PAS_CDB_STR</td>
<td>Convert data to CData_Bool (Digital Structuralization)</td>
</tr>
<tr>
<td>PAS_CDB_DESTR</td>
<td>Convert CData_Bool to data (Digital De-structuralization)</td>
</tr>
<tr>
<td>PAS_BCD_CI16</td>
<td>Convert 16 Booleans to a BCD value</td>
</tr>
<tr>
<td>PAS_BCD_CO16</td>
<td>Convert a BCD value to 16 Booleans</td>
</tr>
</tbody>
</table>

**Description of the Data Conversion Utilities:**

PAS_CDR_TO_PV and PAS_CDR_DESTR are nearly the same, except for the following:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAS_CDR_TO_PV</td>
<td>Outputs the data in Engineering Units</td>
</tr>
<tr>
<td>PAS_CDR_DESTR</td>
<td>Outputs normalized data (0 ~ 100%)</td>
</tr>
</tbody>
</table>

PAS_PV_TO_CDR and PAS_CDR_STR are nearly the same, except for the following:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAS_PV_TO_CDR</td>
<td>Input data is in Engineering Units</td>
</tr>
<tr>
<td>PAS_CDR_STR</td>
<td>Input data is normalized data (0 ~ 100%)</td>
</tr>
</tbody>
</table>
2.6.5.2 Converting Boolean switches to PAS_POU switches

Many switches used in PAS_POU function blocks are Integers and not Booleans.

- Therefore, when an ON/OFF type value is used to drive a PAS_POU switch, it must first be converted through a PAS_ADD* Function.

- When converting a PAS_POU switch to a Boolean value, it must be converted through a PAS_SUB* Function.
The functions available for converting switches are as follows:

<table>
<thead>
<tr>
<th>Function</th>
<th>Input</th>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAS_ADDINT</td>
<td>Integer</td>
<td>Integer</td>
<td>add 1 to the result</td>
</tr>
<tr>
<td>PAS_ADDSW</td>
<td>Bool</td>
<td>Integer</td>
<td>convert to integer and add 1 to the result</td>
</tr>
<tr>
<td>PAS_SUBINT</td>
<td>Integer</td>
<td>Integer</td>
<td>subtract 1 from the result</td>
</tr>
<tr>
<td>PAS_SUBSW</td>
<td>Integer</td>
<td>Bool</td>
<td>convert to boolean and subtract 1 from the result</td>
</tr>
</tbody>
</table>

The types of switch inputs in PAS_POU blocks that require conversion are as follows:

<table>
<thead>
<tr>
<th>Terminal</th>
<th>Description</th>
<th>Value</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTLOCK</td>
<td>Mode Change Interlock Switch</td>
<td>1 = OFF</td>
<td>ILOCKSW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = ON</td>
<td></td>
</tr>
<tr>
<td>TSI</td>
<td>Tracking Switch</td>
<td>1 = OFF</td>
<td>TSW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = ON</td>
<td></td>
</tr>
<tr>
<td>POS_IN</td>
<td>Selector Switch Position</td>
<td>1~3 or 4 or 9</td>
<td>POS</td>
</tr>
<tr>
<td>POS_OUT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRG_RST</td>
<td>Program Reset</td>
<td>1 = OFF</td>
<td>PRGRSTSW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = ON</td>
<td></td>
</tr>
<tr>
<td>PRG_END</td>
<td>Program End</td>
<td>1 = OFF</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = ON</td>
<td></td>
</tr>
<tr>
<td>SWI</td>
<td>Answerback Bypass Switch</td>
<td>1 = OFF</td>
<td>BPSW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = ON</td>
<td></td>
</tr>
</tbody>
</table>

2.6.5.3 Converting General Data Types

The IEC Functions include type conversion blocks in the “Type conv. FUs” library.

For example, the BOOL_TO_REAL block converts a Boolean to a Real value. See the IEC61131 Programming training manual for more information on these blocks.
2.6.6 Cascade Read-Back Connections

In IEC61131 Function Block programming, data can only flow in one direction in a wire that connects one block to another. However, there are many circumstances where tracking of an output is required by a source block. This is particularly the case with cascading control loops, and analog output handling.

Function Blocks that have some sort of cascade action have read-back terminals:

- RB_IN
- RB_OUT

and these are used for the read-back functionality of the loop.

2.6.6.1 Cascade Control Loops

When PID control loops are connected together in a cascade loop, the output of the secondary controller must be read-back to the primary controller as follows:

![Diagram of cascade control loops]

Note: the above example shows a configuration in which a continuous loop is created. Such configuration is not recommended in the IEC standard, and a warning is displayed when compiling the worksheet. It is better to use variables as shown on the next page.
This also applies across signal selectors, signal distributors, and other intermediary blocks, as the following examples show.

**Example 1 – Cascade Connection across a calculation block:**

![Diagram](image)

**Example 2 – Cascade Connection across a signal distribution or switch block:**

![Diagram](image)

1. Note that for every block a distributor block splits the signal to, there is a corresponding read-back terminal on the input.
2. This example also applies to switch blocks such as the PAS_SW13 where there are multiple outputs.
Example 3 – Cascade Connection across a signal selector block:

Note: if an FFSUM block (PAS_FF_SUM) is used, the RB_OUT terminals relate to the preceding blocks in the following way:

- **RB_OUT1** → Connects to the RB_IN of the block connected to the CAS_IN terminal.
- **RB_OUT2** → Connects to the RB_IN of the block connected to the IN terminal.
2.6.6.2 Analog Outputs

When connecting to analog outputs (PAS_AO_ANLG), there is a two way connection between the output function block and the device label (i.e. the hardware output point) in which the actual output is fed back to the output module for checking purposes.

When a device label is assigned to an output, and read-back tag is automatically created by the system. If, for example, the device label is CV100, then the read-back tag is called CV100_RB. This is connected to the output function block as follows:
2.6.7 Reporting Alarms to the HMI

Stardom has an alarm and event message handling function that does not require programming or setup in the FCX. In addition, it is possible to create an alarm message in the FCX that is reported to the VDS as a standard alarm message. This is an annunciator type function block and can be used to initiate an alarm due to a digital input or PLC input.

- The function used to create the alarms is: **PAS_MSG_UPRCALM**
- and this is in the library: **SD_FIELD_PF**

**Function** – when the ONOFF input (Bool type) goes ON, the message connected to the USER_STR input is transmitted as an alarm to the VDS. The input to the EN input must be TRUE for the function to be active.

**Example 1:**

When the digital input PSH100 goes ON, then an alarm is raised in the VDS with the message ‘Inlet Pressure Hi’ and the Object name, PSH100.

**Example 2:**

Certain global variables have values embedded in them. These can be found in the Status Definition category of the Global Variables file, and are described in more detail in section 2.4.5. In this example, the global variable GM_ALRM_HH (which contains the string ‘HH’) is used:

(Note: see section 2.4.2 for an explanation of the * _BOOL variable.)
2.6.8 Inter-FCX Communication (SD_FCXCOM_LIB)

An FCX can receive and send data with up to 15 other FCXs over ethernet using the SD_FCXCOM_LIB functions.

There are two types of communication functions:

- **Unconfirmed communications** – where the sender does not receive notification that the data has been received by the receiver.

- **Confirmed Communications** – where the sender receives notification that the data has been received correctly by the receiver. This requires more memory than unconfirmed communications.

The function blocks available for communicating data between FCXs are as follows:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNECT</td>
<td>Connection block for Communications</td>
</tr>
<tr>
<td>READ_1V</td>
<td>Confirmed – Read 1 Value</td>
</tr>
<tr>
<td>READ_5V</td>
<td>Confirmed – Read 5 Values</td>
</tr>
<tr>
<td>READ_10V</td>
<td>Confirmed – Read 10 Values</td>
</tr>
<tr>
<td>WRITE_1V</td>
<td>Confirmed – Write 1 Value</td>
</tr>
<tr>
<td>WRITE_5V</td>
<td>Confirmed – Write 5 Values</td>
</tr>
<tr>
<td>WRITE_10V</td>
<td>Confirmed – Write 10 Values</td>
</tr>
<tr>
<td>URECV_1V</td>
<td>Unconfirmed – Read 1 Value</td>
</tr>
<tr>
<td>URECV_5V</td>
<td>Unconfirmed – Read 5 Values</td>
</tr>
<tr>
<td>URECV_10V</td>
<td>Unconfirmed – Read 10 Values</td>
</tr>
<tr>
<td>USEND_1V</td>
<td>Unconfirmed – Write 1 Value</td>
</tr>
<tr>
<td>USEND_5V</td>
<td>Unconfirmed – Write 5 Values</td>
</tr>
<tr>
<td>USEND_10V</td>
<td>Unconfirmed – Write 10 Values</td>
</tr>
</tbody>
</table>
To simplify the communication programming around the CONNECT block, a set of function blocks have been created by Yokogawa as follows:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD_CFCX_CONNECT_EX</td>
<td>Use in place of the CONNECT function</td>
</tr>
<tr>
<td>SD_CFCX_MKPARTNER</td>
<td>Sets up a string to pass to the CONNECT block.</td>
</tr>
<tr>
<td>SD_CFCX_PULSE</td>
<td>Used in the TRIG block.</td>
</tr>
<tr>
<td>SD_CFCX_TRIG</td>
<td>Sets up communication time interval triggers for communications.</td>
</tr>
</tbody>
</table>

Since MKPARTNER and PULSE are used inside CONNECT_EX and TRIG, these are not used by the programmer as such.

**SD_CFCX_CONNECT_EX**

Contains the CONNECT block to connect with a related CONNECT block in another FCX.

- **EN_C**: if the input is TRUE, then the connection is made.
- **PORT**: set to a number between 1 and 16. The same number should be set in the port of the block in the other FCX.
- **HOST**: the IP address or hostname of the FCX to which the connection is being made.
- **ID**: Returns a valid ID connection value for use by the Send and Receive blocks.
- **VALID**: Returns a TRUE if there is a successful connection with the other FCX.
- **ERROR**: Returns a TRUE if there is an error in communicating to the other FCX.
- **STATUS**: Returns an error code if there is an error in communication.

**SD_CFCX_TRIG**

Sets up a trigger for the write block, according to a set time interval for normal communications, and a time interval where there is an error in communications.

- **IN_VALID**: If this input is TRUE, then the trigger function is enabled.
- **IN_ERROR**: If this is TRUE (i.e. there is an error in the Send communication), then the trigger time is based on IN_TIME2.
- **IN_TIME1**: Trigger time for normal communication.
- **IN_TIME2**: Trigger time in the event of an error.
- **Q**: Trigger pulse output.
2.6.8.1 Unconfirmed Communications

Unconfirmed communications require a minimum of four blocks to send or read a data value from one FCX to another:

- **CONNECT** – sets up the connection between the FCXs, with IP address specification. (2 required). For programming purposes, the SD_FCX_CONNECT_EX block and the SD_FCX_TRIG block are used instead of the CONNECT block.
- **UREAD_*V** – reads data from a USEND block in the other FCX.
- **USEND_*V** – writes data to a UREAD block in another FCX.

where ‘*’ is 1, 5 or 10, depending on the number of data to be read or written.

Note that USEND and UREAD must be connected in pairs between the FCXs.
An example of how the communications is setup is shown below:

Function:

1. The CONNECT_EX port sets up communication with FCX of IP address 172.168.0.2 on Port 1.
2. If the link is made, the VALID terminal goes TRUE, and this runs the trigger block.
3. If there is no error from the Send block, the Q terminal pulse ON every second (according to the time set in IN_TIME1).
4. When the USEND block receives the trigger in the REQ terminal, it sends the value in SD_1 to the FCX defined in the ID terminal.
5. The DONE goes to TRUE on successful communications.
6. If there is an error in communication, the ERROR terminal goes TRUE, and this is fed-back to the trigger block which triggers the send at the time set in the IN_TIME2 terminal.
7. The URECV block is not triggered but receives data from the FCX identified in the ID terminal when it is enabled by the EN_R terminal. The EN_R terminal is usually linked to the VALID terminal of the CONNECT_EX block.

The terminal parameters for the SEND/RECEIVE blocks are as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Integer value that identifies the IP address and Port number of the FCX it is communicating to. This is set by the CONNECT block.</td>
</tr>
<tr>
<td>R_ID</td>
<td>A literal string parameter. The same string must be set in the associated block in the other FCX.</td>
</tr>
<tr>
<td>EN_R/REQ</td>
<td>Enables the Read/Write function.</td>
</tr>
<tr>
<td>RD*/SD*</td>
<td>Read and Write terminals for connecting data of any data type.</td>
</tr>
<tr>
<td>NDR/DONE</td>
<td>Returns TRUE when Read/Write successfully completed</td>
</tr>
<tr>
<td>ERROR</td>
<td>Returns TRUE if there is a communication error</td>
</tr>
<tr>
<td>STATUS</td>
<td>Returns an error code (INT) if there is a communications error.</td>
</tr>
</tbody>
</table>
2.6.8.2 Confirmed Communications

Confirmed communications require a minimum of three blocks to send or read a data value from one FCX to another:

**CONNECT** – sets up the connection between the FCXs, with IP address specification. (2 required). For programming purposes, the SD_FCX_CONNECT_EX block and the SD_FCX_TRIG block are used instead of the CONNECT block.

**READ_*V** – reads data from another FCX.

**SEND_*V** – writes data to another FCX.

where ‘*’ is 1, 5 or 10, depending on the number of data to be read or written.

In this case, Read/Write pairs are not required.

**PDD** - Data that is being read from or written to are local or global variables and must have the **PDD** (Process Data Directory) specification checked in the data dialog box.
An example of how the communication is setup is shown below:

The functional description is the same as that given for unconfirmed communications. However, note the following differences:

- There is no corresponding READ or WRITE block in the other FCX.
- For each parameter being read or written, there are two parameters to be set up: VAR_*, and SD_*/RD_*.

SD_* and RD_* are the same as those described in the section on unconfirmed communications, and contain the value to be read or written.

VAR_* - because there is no associated READ/WRITE block in the other FCX, the data point must be defined to which the value set in SD/RD is to be read/written. In other words, the VAR_* input connects to a string variable that contains the name of the variable in the other FCX to which the data is being written.
Defining the VAR input:

Assume that there are two FCXs called FCX01 and FCX02. In FCX01, define local or global variables for data to be read and written to by FCX02:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Usage</th>
<th>Description</th>
<th>Add...</th>
<th>Init</th>
<th>Retain</th>
<th>FDD</th>
<th>OPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCX01_READ_POINT</td>
<td>CData_REAL</td>
<td>VAR_GLOBAL</td>
<td>Value to be r...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FCX01_WRITE_POINT</td>
<td>CData_REAL</td>
<td>VAR_GLOBAL</td>
<td>Value to be</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The data type for these will be set according to the function blocks they are to be connected to, and will be the same as the data types of the SD and RD variables. Make sure that PDD is selected.

In FCX02 - in the worksheet define a STRING variable (e.g. WT_DATA1_NAME), and set its initial value to the associated global variable (e.g. '@GV.FCX01_WRITE_POINT').

The function of the WRITE block is to take the WR_DATA_1 value in FCX01 and put it into the FCX02_WRITE_POINT global variable in FCX02.

The function of the READ block is to take the FCX02_READ_POINT global variable in FCX02 and put it into the RD_DATA_1 value in FCX01.

@GV is used as an identifier to specify a global variable.
2.6.7.3 Programming FCX Read/Write functions in Logic Designer

Once the read/write program is setup, the variables can be used by other blocks.

**Example 1** – writing an analog input to another FCX using confirmed communication:

![Diagram of WRITE_1V block](image)

The analog input is loaded into parameter “WR_DATA_1”. The WRITE_1V block writes this data to the other FCX (defined in the CONNECT block not shown here).

**Example 2** – reading an input from another FCX using confirmed confirmation:

![Diagram of READ_1V and PAS_PVI blocks](image)

The input is read from the other FCX and loaded into the RD_DATA_1 variable by the READ_1V block. It is then input into the PAS_PVI block.

A read/write template for both Confirmed and Unconfirmed read/write functions are available in the **Example E** project and may be copied into a user POU for your use.
2.7 Debugging Tools

**Debug** mode is enabled in Logic Designer by selecting Debug mode from the Online menu. This only works when the FCX is communicating to the PC, and the project in the Logic Designer is the same as the one in the FCX. When running, data can be viewed on the program sheets, and Logic Analyzer can be run.

When running in debug mode, process I/O can be simulated using the **softwiring** facility. This provides loopback from outputs to inputs, and simple delays and lags can be specified.

The **Logic Analyzer** is a powerful tool for recording values of variables over a certain time interval which the user has to determine by a entered number of sample cycles. Using this feature you can check the behavior of your PLC program by recording variables (while the PLC runs) and displaying them graphically in the Logic Analyzer window. It is possible to display several curves of variable values together depending on their trigger conditions and other settings.

All recorded values and settings of the Logic Analyzer are stored automatically with the project. Additionally it is possible to export the recorded data into an ASCII file with the file extension *.csv.

The system supports the use of the Logic Analyzer with one or several resources. Each resource has a separate Logic Analyzer worksheet. A new resource is added or deleted in the Logic Analyzer window after compiling the project.

Note: Recording values is only possible in online mode.

### 2.7.1 Running Debug Mode:

The pre-requisites for running Debug Mode are:

1. The project in the Logic Designer is the same as the one in the FCX
2. The FCX is in run mode (see section 2.3.4.2)

It is running successfully if there is a green line displayed at the bottom of Logic Designer.
2.7.2 Softwiring Procedure

The softwiring tool allows Input and Outputs to be connected together, thereby simulating the process I/O. This tool actually creates a program POU (ST type) which must be instantiated as a task to operate. In addition, a global variable (GS_NFIO_DISCONNF) must be set to TRUE to disconnect the I/O variables from the actual I/O.

The easiest way to setup softwiring is using the I/O Simulation Wizard, by clicking on the Wizard button in the SoftWiring dialog. However, the procedure for manual wiring is given below so that the user is aware of how the simulation works. The Wizard dialog is shown on the next page.

**Procedure:**

1. Double click on Softwiring in the FCX resources folder.
2. Type in the POU name to be created.
3. Enter the Input and Output I/O names as already defined in the DeviceLabelDefinition.
4. Select the Wiring Logic (SD_WIRE_DTANLG_ANLG for analogs). Other settings are also available in the Advanced dialog.
5. Click OK. A Program POU is created. It is a Structured Text program using the Functions selected in the Wiring Logic box (SD_WIRE_PF library).
6. Add this program to a task in the FCX.
7. Make and Download the program and select Debugging.
8. Go to the Global Variables file, Interface Flags section, and set the variable: GS_NFIO_DISCONNF to TRUE to disconnect the process I/O wiring, enabling the softwiring to work.

Alternatively, from step 1, click on WIZARD and the following dialog appears:
Here the data can be entered with the aid of drop-down menus. When finished, the data is entered into the SoftWiring dialog. Proceed from Step 5 above.
2.7.3 Logic Analyzer Procedures

**Prerequisites:**

*Before calling the Logic Analyzer make sure that:*

- the compiled project is downloaded to the PLC.
- the FCX is in the running state.
- the system is in online mode.
- the system is in Debug mode.

1. Open the Logic Analyzer window. This is in the VIEW menu.

2. Add variables to the Logic Analyzer. Do this by right-clicking on a variable in a program sheet and selecting Add to Logic Analyzer. Note that you must be in debug mode for this to work.

3. If necessary delete/remove variables connected to the Logic Analyzer, by right-clicking on the variable and selecting delete/remove.

4. Open the Logic Analyzer context menu (Online menu or right click on the tab) and select the menu item 'Window Width'. Set the initial window width in the appearing dialog 'Data Window'. The default value is 200 samples.

5. Set the trigger configuration. See the Logic Designer help file for more information.

6. Click on the Icon 'Start recording values' in the toolbar. The Logic Analyzer starts recording values according to the time interval defined in the dialog 'Trigger configuration'.

7. If you want to cancel the recording press the icon 'Stop recording values' in the toolbar. In a valid record process the recording of values stops automatically at the end of the defined sample cycles.

8. Adjust the Logic Analyzer window if required.

9. If you want to delete all curves and listed variables from the Logic Analyzer in one step, select the menu item 'Clear Curves' either in the Logic Analyzer context menu or in the submenu 'Online'. If displayed in the toolbar, you can also use the icon 'Clear Curves'.

10. If you want to capture the visible curves, click on the icon 'Capture Curves' in the toolbar.

11. If you want to export recorded data into a *.csv file select ‘Export Data’ from the Online – Logic Analyzer window.
2.7.4 The Watch Window

The watch window can be used to display variables from different worksheets to monitor how these variables work together. The watch window can used to debug elements of user defined data types such as arrays and structures.

In the watch window you can insert and delete variables. There is no limit for the number of variables inserted in the watch window. In addition you can use the watch window to force and overwrite variables. This is performed by marking a variable in the watch list and choosing the menu item 'Debug dialog...' in the context menu of a variable.

How to display/hide the watch window:

- Choose the menu item 'Watch Window' in the submenu 'View' or click on the icon 'Watch window' in the toolbar. Depending on its previous state, the watch window is now visible or hidden.

Note: You can open the Watch Window also by selecting the menu item 'Open Watch Window...' in the context menu of a variable (only possible in online mode).

The watch window contains the following information:

- 'Variable': Displays the variable name. User defined data types such as arrays and structures are marked with a '+' sign. To display the elements of these data types, click on the '+' sign.
- 'Value': Displays the current value of the variable.
- 'Default Value': Displays the default value (if defined).
- 'Type': Displays the data type.
- 'Instance': Displays the instance path where the variable is used. The path always contains the configuration, resource, task, the associated program name and variable name.
The watch window allows the user to manage several pages where every page can be used independently. The individual pages are called by clicking on the sheet tabs at the bottom of the watch window.

How to display the cross references of a variable shown in the watch window:

It is possible to get direct access to cross references from the watch window. For that purpose just mark the desired variable in the watch window. The marked variable is automatically marked in the cross reference window too.

You can insert variables from a variable declaration or from textual and graphical worksheets in the watch window.

How to insert variables in the watch window:

- Mark the variable in the worksheet or variable sheet in online mode.
- Press the right mouse button to open the context menu.
- Choose the menu item 'Add to Watch Window'.
- The variable is inserted in the watch window.

How to remove variables from the watch window:

You can either remove a single variable from the watch list or clear the whole list:

- To remove a single variable from the watch list, mark the variable to be deleted and press the key <DEL>. Alternatively you can open its context menu using the right mouse button and select the context menu item 'Delete'.

Note: This is also possible for a single element within an user defined structure or array.

- To remove all variables open the context menu of any variable in the Watch Window and select the menu item 'Clear'.
2.8 HTML Applications

The FCX has a built in web server which can be used to provide operator type displays directly from the FCX without the use of an HMI.

A standard web page can be generated using Frontpage or other editor, and downloaded using FTP. The web page requires Java Objects to connect to the data in the FCX.

To connect to the FCX by FTP, using an FTP package, log into the FCX by using its IP address. Default login parameters are:

- Username: stardom
- Password: YOKOGAWA

Download the web pages into the WWW folder. The main web page must be called INDEX.HTM.

To view the web pages in the FCX, type the IP address of the FCX into a web browser. The Index page will appear.

Java Objects required to be used with the web pages are covered in a separate course.
## 3.1 Introduction

3.1.1 Overview of Functions ................................................................. 4
3.1.2 Structure of the VDS ................................................................. 5

## 3.2 General Setup

3.2.1 Installation .................................................................................. 7
3.2.2 User Accounts ........................................................................... 8
3.2.3 VDS System Launcher ................................................................. 9
3.2.4 Communications Setup ............................................................. 10
3.2.5 License Management ................................................................. 11
3.2.6 System Configuration ............................................................... 12
3.2.7 Backup Tool ............................................................................. 14

## 3.3 Object Builder

3.3.1 Using the Object Builder ......................................................... 15
3.3.2 Importing the FCX Tag Database ............................................ 16
3.3.2.1 Instance Names ............................................................... 16
3.3.2.2 Object Classes ............................................................... 17
3.3.2.3 Object Import Function .................................................. 17
3.3.2.4 Writing to Switches .......................................................... 19
3.3.3 Creating Control Groups ......................................................... 19
3.3.4 Creating Control Objects Manually ....................................... 20
3.3.5 Creating An I/O Object ............................................................ 21
3.3.6 Running Data Server Functions ............................................. 22
3.3.7 Procedures .............................................................................. 23
3.3.7.1 Importing and Modifying the Tag Database ..................... 23
3.3.7.2 Running VDS Function .................................................... 26

## 3.4 Graphics Designer

3.4.1 General .................................................................................... 27
3.4.2 Primitives ............................................................................... 28
3.4.3 Display Objects ....................................................................... 28
3.4.4 Function Objects ..................................................................... 29
3.4.5 Graphic Modify ....................................................................... 31
3.4.6 Linked Parts ........................................................................... 32
3.4.7 HMI Deployment Tool ............................................................. 34

## 3.5 Historian and Trend Function

3.5.1 Structure of the Historian ....................................................... 35
3.5.2 Function of the Historian ......................................................... 36
3.5.3 Procedures ............................................................................. 38

## 3.6 Report Function

3.6.1 General ................................................................................... 40
3.6.2 Report Functions .................................................................. 41
3.6.3 Data Collection and Processing Function ............................... 42
3.6.4 Batch Reporting Function ...................................................... 43
3.6.5 Selecting the Report Data ....................................................... 44
3.6.6 Closing Processing Configuration ......................................... 45
Section 3. VDS Functions

3.6.7 Report Function Setup

3.7 Message Function

3.8 Security Function

3.9 HMI Client/Server Functions
   3.9.1 HMI Server Configuration
   3.9.2 Operator Display Function
3.1 Introduction

3.1.1 Overview of Functions

The Versatile Data Server provides two main functions:

- Data Server Function
- HMI Server Function

The Data Server function reads data from control devices, such as FCX or FA-M3 PLC, and holds this data in the form of Control Objects. This data may be accessed using OPC by the HMI Server function, or some other application.

The HMI Server reads data from the Data Server using OPC, and serves operator displays in the form of web pages to a web client, i.e. PCs running Microsoft Internet Explorer.

*The functions available in the VDS are as follows:*

<table>
<thead>
<tr>
<th>HMI Client Function</th>
<th>Object View Display</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Graphic Window Display</td>
</tr>
<tr>
<td></td>
<td>Log On</td>
</tr>
<tr>
<td></td>
<td>Trend Displays</td>
</tr>
<tr>
<td></td>
<td>Message Display</td>
</tr>
<tr>
<td></td>
<td>Audible Message</td>
</tr>
<tr>
<td>HMI Server Function</td>
<td>Web Server</td>
</tr>
<tr>
<td></td>
<td>Session Management</td>
</tr>
<tr>
<td></td>
<td>Data Access</td>
</tr>
<tr>
<td>Data Server Function</td>
<td>Data Server with Control Objects</td>
</tr>
<tr>
<td></td>
<td>I/O Drivers</td>
</tr>
<tr>
<td></td>
<td>- FCN/FCJ</td>
</tr>
<tr>
<td></td>
<td>- FA-M3</td>
</tr>
<tr>
<td></td>
<td>- OPC Server</td>
</tr>
<tr>
<td></td>
<td>Message Management</td>
</tr>
<tr>
<td></td>
<td>Task Management</td>
</tr>
<tr>
<td></td>
<td>Database (ISAM)</td>
</tr>
<tr>
<td></td>
<td>Security Management</td>
</tr>
</tbody>
</table>
3.1.2 Structure of the VDS

The Data Server comprises the I/O Driver to field device using I/O Objects, the Data Server Engine using Control Objects, and an OPC Server that makes the data available to the HMI server. A VBA Engine provides access to Control Objects by Visual Basic programs.

This system is configured using the Object Builder.
The **HMI Server** comprises a **Data Access Service** for reading data from the Data Server, and a **Web Server** for serving Graphic Applications to HMI Clients.
3.2 General Setup

3.2.1 Installation

1. Install Windows2000 and Service Pack 2.
2. Create a user account for VDS. (e.g., stardom)
3. Add “Administrator” authority to this account.
4. Logon as “Administrator” and install VDS software.
5. Note: JAVA plug-in must be installed.
6. Logon as an user account for VDS (stardom) and register licenses with “license management tool” (see section 3.2.3).
7. Perform Initial Setup with the Initial Setup tool. A reboot is required.

   i) Click on Setup Folder and OK.
   ii) Select a work folder (the default is recommended)
   iii) Click SET. The setup parameters will be written to the System Registry.

8. VDS will now start whenever the VDS account (stardom) is logged on.
### 3.2.2 User Accounts

As mentioned above, a user account for the VDS must be installed. It is recommended that this user account is called “stardom”. This account must have Administrators privileges. In addition, this account must also be a member of the **VDS Manager** group.

When the VDS software is installed, the following groups are created in Windows:

- VDS Manager
- VDS Operator1 ~ 7

These can be viewed in


For Operators who are logging into the system, they can be assigned as members of a VDS Operator Group. See **section 3.8** for details on user access configuration for VDS Operators.

The system also creates a user called **ASTMACSystem** on installation. This should be a member of **Administrators** and **VDS Manager** groups. **Do not modify this ASTMACSystem user and do not use this account in normal operation.**
3.2.3 VDS System Launcher

All VDS configuration functions are available through the VDS System Launcher. This is accessed through:


or by double clicking on the Yokogawa VDS icon on the desktop:

![Yokogawa VDS icon]

This brings up the VDS System Launcher:

![System Launcher window]

A summary of the tools available from the launcher are as follows:

- **Development Builder** – Contains all the functions for building the operational functions, such as Object Builder, Graphic Builder.

- **Mode Change** – allows switching between development, operations and shutdown modes.

- **Support Tools** – Contains system maintenance tools.

- **System Configuration** – for initial setup of the system (see 3.2.6).

- **System Management** – for backups and additional licenses.
3.2.4 Communications Setup

Section 2.2.4 covers IP address setup for the PC in detail. In summary, IP addressing in the PC should be setup as follows:

If the Virtual IP address in the FCX is \texttt{xxx.yyy.0.zzz}, then set the following IP addresses for the PC:

- \texttt{xxx.yyy.0.nnn}
- \texttt{xxx.yyy.1.nnn}
- \texttt{xxx.yyy.2.nnn} (if dual ethernet)

Once this has been set, the \textbf{VDS – FCX Connection Setting} must be configured. In this, the Virtual IP address is identified from the above mentioned addresses for use by the VDS.

Access through:

If there is a redundant ethernet, click on the Redundancy Service tab and check “Enable Redundant Network Service”.

The PC must be rebooted after these parameters are set.

The Time Service function is described in section 3.2.6.
3.2.5 License Management

The licenses that have been installed can be confirmed through the License Manager. In addition, new licenses can be installed.

From System Management, double-click on License Management. This displays the current license information, including a list of all software packages currently loaded.

To register an additional license, click on License Registration. Select license type (usually License file) and click NEXT. Select the file to be installed.
3.2.6 System Configuration

Other setup parameters in System Configuration include:

![System Configuration Window]

Runtime File Setting – Used for setting:

- **Data Server Configuration File** – Select the runtime project file, and the working files location.
- **Message Configuration File** – Select the runtime message file, and the working files location.

**Data Server Name Setting** – set the name of the Data Server (database) and its location (PC). More than one Data Server can be defined (currently a maximum of 4).

**HMI Configurator** – See section 3.6.1 for details on setting up the HMI Server.

**FCN FCJ Connection Setting** – See section 3.2.4 for details on communications setup.
**Process Configurator** – for registering optional packages (such as Trend and Reporting packages) and for User applications (such as a Visual Basic program) that uses the VDS database.

To register the Trend and Report functions, click on browse and select the following folder:

```
```

and select File Type as EXE files.

Select the following files:  
**HistCollection**  
**yfsReport**

For each process:

- Increment the process number.
- Check the Enable box.
- Select Normal for Priority
- Select Normal for Window style

Once the file and parameters are selected, click the ADD button. If a change is made to a configuration, click the UPDATE button. When finished, SAVE and EXIT.
3.2.7 Backup Tool

The Backup Tool allows data to be backed-up and restored to removable disk. The following information can be backed-up:

- Configuration information
- Logs information
- Application information
3.3 Object Builder

3.3.1 Using the Object Builder

The Object Builder links the tag instances in the FCX, FAM3, or other devices (such as a foreign PLC, or OPC gateway) to the Data Server. The Object Builder has the following functions:

- Import and build Control and I/O Objects
- Run/Stop VDS
- History Builder
- Report Builder
- Security Builder

The important terms in the Data Server are:

- **Control Object** – links to an instance of a variable in an FCX or other device.
- **I/O Object** – is a device driver that specifies the protocol of the device from which the variable instances are being read (or written).

For Stardom FCX, Object Builder can automatically generate the Control Objects through the import tool. Control Objects can also be manually created for other devices.

Object Builder has an explorer type interface with folders in the left hand window that relate to the components of the system:

- **Group Folders** – for each device, there is a group that contains the control objects for the device. There is at least one group called ‘MainGroup’ and this cannot be deleted.
- **I/O Object Folder** – device drivers for FCX, FAM3 and other devices.
3.3.2 Importing the FCX Tag Database

3.3.2.1 Instance Names

When the FCX database is imported, the Object Builder creates a set of Control Objects of the PASPOU blocks based on the instance names of the variables. An instance name is built as follows:

```
TaskNameFBNameVariableName....
```

**Example 1:** if tag ‘FIC100’ exists in a program in task name ‘Task1’ (i.e. object address is Task1.FIC100), then the Control Object name will be

```
Task1FIC100
```

**Example 2:** if tag ‘FIC100’ exists in a Function Block called ‘Level’ in task name ‘Task1’ (i.e. object address is Task1.Level.FIC100), then the Control Object name will be:

```
Task1LevelFIC100
```

1. The Variable Name may in fact be a Function Block, as the above two examples show.

2. Note that the maximum number of characters allowable in the object name is **16 characters**. If this is exceeded, then characters are deleted from the **left** of the name.

3. If the task name starts with an underscore ‘_’, then it will not be included in the name of the object. In example 1, if the task name was ‘_Task1’, then the object name would be FIC100.

4. Note that the object name can be changed during the import process, but once created, cannot be changed.

5. What can be a Control Object:
   - Only PASPOU Function Blocks can be Control Objects.
   - User defined function blocks cannot be Control Objects.
   - Variables and I/O can be control objects, but variables must be created manually.
   - However, Data Structures (such as CData_Real) cannot be control objects.
### 3.3.2.2 Object Classes

*Control Objects belong to one of four classes:*

<table>
<thead>
<tr>
<th>Object Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Tag Object</td>
<td>An input/output in the device (FCX or other)</td>
</tr>
<tr>
<td>FCN/FCJ Object</td>
<td>A function block in the FCX</td>
</tr>
<tr>
<td>Process Tag Object</td>
<td>A function block. For connecting to foreign devices.</td>
</tr>
<tr>
<td>Application Support Object</td>
<td>Objects related to system parameters.</td>
</tr>
</tbody>
</table>

Note that during the import process, the object class cannot be viewed or edited. The class is automatically assigned as FCN/FCJ Object or Device Tag Object. However, when creating an object manually, the class must be assigned to the object.

### 3.3.2.3 Object Import Function

During the import process, the builder reads the ADLST.CSV file for the FCX. This file exists in the project for the FCX as follows:

```
\ProjectName\C\Configuration\R\FCXnn   (where nn = the FCX number)
```

This file is the database for the FCX and thus all data for that FCX can be imported.

*The import table:*
The following settings can be modified:

- **Import** – Select whether this object is to be imported.
- **Group Name** – Select which group the object will be imported into.
- **Object Name** – The Object Name has been automatically created as described above. Here this value can be changed. Once imported, this name cannot be changed.
- **Type** – The type associates the object with a function block or I/O type. See below for more information on this.
- **Comment** – a comment describing the object can be entered. This can be seen in the object viewer and on the module display in the HMI.
- **Using VBA** (not shown above) – this can be accessed by reducing column width of other fields. Select whether VBA is used. This is a global change to all objects.

Note that only variables marked as OPC type in the Logic Designer will be picked up by the import utility. This applies to variables only and not Function Blocks.

Once created, the object can be modified by double clicking on the object name. Note that the object name cannot be changed. However the address and certain object attributes can be. These can be found in the INTRINSIC tab of the properties dialog of the Control Object.
3.3.2.4 Writing to Switches

Many variables in the FCX (or other device) are internal switches, or flags. When imported, these are automatically designated as DI’s, that is, Digital Inputs. This means that for the Data Server, they are read only, and cannot be set from a graphic.

To enable writing to switches, another control object for each one needs to be manually created in which the object is designated as a DO. Alternatively, the control object can be designated as DR which allows read/write access to the object.

For example, a switch of name P1DUTY will have 2 Control Objects associated with it:

- P1DUTY – Type: DI (Class = Device Tag)
- P1DUTY_DO – Type: DO (or DR) (Class = Device Tag)

In the graphic that is developed in the HMI, a pushbutton or other data setting object would write to the P1DUTY_DO control object.

3.3.3 Creating Control Groups

When importing an FCX database, a Control Group is created automatically. The name of the Control Group is the name of the Resource name given to the FCX in Logic Designer (usually FCX01, etc).

During the import process, it is possible to assign Control Objects to other Control Groups. In addition, manually created Control Objects can be assigned to any Control Group.

Control Groups must be created before Control Objects are created or imported.
3.3.4 Creating Control Objects Manually

In some cases such as that mentioned above, or when linking to other field devices for which no import utility is available, the control object must be created manually.

A control object can be created in any Group (folder), and linked to any IO Object (driver). This includes OPC tags, PLCs, etc.

*The data that must be set is as follows:*

- **Name** – Control Object name
- **Class** – Class of the Object
- **Type** – Object type

I/O Information

- **Name** – Name of the I/O Object (Device Driver)
- **Address** – The name of the variable in the source device
- **Scanning Method** – Scan at which the Control Object reads/writes to the variable in the source device.

In addition, object attributes can be set in the INTRINSIC tab, such as scan rate, alarm properties, etc.
3.3.5 Creating An I/O Object

An I/O Object is a driver for a field device. The drivers available are:

- Stardom Controller (FCN/FCJ)
- FA-M3 PLC (Ethernet, Serial, Modem)
- OPC
- MELSEC (Ethernet and Serial)
- SYSMAC
- EZSocket
- DARWIN
- DAQSTATION
- M&C Power Monitor

When an FCX controller database is imported, the I/O Object is automatically created. The properties of the Object are accessed by double-clicking on it.

In the OTHERS tab, make sure that the IP address is set in the Resource Name. By default, the I/O Object name is the Resource Name (e.g. FCX01), but unless this is specified in the LMHOSTS file of the PC, it should be replaced with the IP address.

This is because the LMHOSTS file in Windows 2000 is a look-up table that relates a host name to an IP address.
3.3.6 Running Data Server Functions

The Object Builder also functions as the control panel for Data Server operations.

The Data Server can run in three Modes:

- Design/Development
- Debugging
- Runtime/Operation

The mode is set either through the TOOLS menu of the Object Builder, or through the Stardom icon in the system tray.

Note that the system tray utility allows switching between Development and Operation, but not Debugging. This can only be done through the Object Builder, and only when the system tray utility is set to Development.

When the Data Server mode is Runtime or Debugging, it starts reading values through the Control Objects. The HMI server function runs automatically as a Windows service after the PC is turned on.
3.3.7 Procedures

3.3.7.1 Importing and Modifying the Tag Database

Procedure for importing the FCX database

1. In the Object Builder, select FILE → IMPORT SETTINGS

2. Under ‘File Type’ click on SELECT. Make sure that both Control Objects and IO Objects are ticked. Click OK.

3. Select the type of CSV file (STARDOM ADLST), and make sure that ‘Use Previous Settings’ is not selected. Click OK.

4. Select the file location. This will be in the FCX file for the project as this example shows. Select the ADLST file and click OK.
5. The list of Control Objects for that FCX appears, and Object Names, Group Names and Type can be edited. The check box on the left is used to specify whether the object is imported or not. When finished, click OK, and the objects are imported into the appropriate groups.

Procedure for modifying the FCX database

Once the Control Objects are imported, they can be modified by double clicking on the object. Here the address from where the data is read/written can be changed.
Procedure for creating a Group

1. Click on ‘Workspace’
2. Select PROJECT \(\rightarrow\) Add Group.
3. The Group will appear as a folder under Workspace.

Procedure for manually creating a control object

1. Select the group in which the object will be located.
2. Click on PROJECT \(\rightarrow\) ADD CONTROL OBJECT

3. Give the object a name, and select its class and type.

4. The same box appears as when modifying an existing object. Select the IO Object (or driver) the object will be reading/writing the data from/to, and the address of the variable or function block in the field device.
3.3.7.2 Running VDS Function

- To run the VDS function through Object Builder:

  Click on the TOOLS menu and select MODE CHANGE. The following window appears:

  - The Project must be saved before Debugging or Runtime can be selected.
  - When Debugging is selected, a Visual Basic window appears. This can be closed down if not needed. Debugging is selected during system design for testing the applications.

- To run the VDS function though the System Tray:

  Double-click on the Stardom icon in the System Tray. The following window appears:

  This window allows switching between Operation (Runtime) and Development (Design) Mode, or shutting down of the Data Server functions. Debugging mode (above) can only be run when in Development mode.
3.4 Graphics Designer

3.4.1 General

See the CS3000 Instruction Manual 33S04H10-01E, Section 6, with full procedures for drawing graphics, and 33S01B30-01E, Section F12 for details on graphic builder functions.

The main differences between creating a graphic in Stardom relate to the way Control Objects are addressed in graphic objects, and this is discussed below.

Also, a graphic is embedded in a web page, and this web page can be modified using a standard html editor.

**Capacity** – a graphic page can have a maximum of 200 primitives.
3.4.2 Primitives

A display object such as a bargraph display, or a data value display references to a Control Object in a Group as defined in the Object Builder.

- Specifying data in display object on a graphic:

The format for linking to a Control Object for display is as follows:

@Data(“Group_Name.Control_Object_Name.Variable”)

For example, to display the PV of Control Object ‘FIC100’ in the FCX01 Group:

@Data("FCX01.FIC100.PV")

- Data character display
- Data rectangle bar display
- Data arrow bar display
- Data circle bar display
- Trend
- Message
- Push button
- Touch target

- 2-dimensional graph
- Radar chart
- Step form graph
- Stick graph
- User definition line graph
- Line graph
3.4.4 Function Objects

A function object is a Touch Target or Pushbutton in which an action is carried out when it is clicked on.

There are several functions available:

- **Open URL** – call up another web page
- **Open Monitoring or Logon Window** – call up another graphic page
- **Setting** – write a pre-defined data value to a Control Object
- **Call Data Input Window** – call a dialog for entering a value to be written to a Control Object
- **Call Menu Dialog** – call a menu of values to write to a Control Object
- **Print** – print the graphic page

Specifying data in a function object on a graphic:

The format for linking to a Control Object depends on the function.

**Open Monitoring or Logon Window**

There are several types of windows that can be opened:

- Graphic Index Window – calls the list of graphics
- Graphic Window – calls a specified graphic
- Object View Index Window – calls the list of Control Objects
- Object View Window – calls a Control Object as a Faceplate
- Logon Window
- Logoff Window

For the Graphic Window, the syntax for Window Name is:

“Window_Name”

For the Object View Window, the syntax for Window Name is:

“object=Group_Name.Object_Name”

**example**: “object=FCX01.FIC100” calls the FIC100 PID faceplate

For the other window types, it is not necessary to specify a window name.
Setting Functions

The setting functions are:

- Setting
- Call Data Input Window
- Call Menu Dialog

For these, the syntax for the Data is:

“Group_Name.Control_Object_Name.Variable”

For example:

“FCX01.PSTART_DO.CV”

**Note:** to know which variable to use for a Control Object, the nature of the Data Structure must be understood. Refer to Section 2.4.2 for more information on Data Structures for Function Blocks.
3.4.5 Graphic Modify

Objects (primitives) can be modified according to process conditions. The type of modifiers available are:

- Color Change
- Alarm blinking
- Conditional blinking
- Make primitive invisible
- Modify string (if primitive is text)
- Invert String (if primitive is text)

A total of 8 modifies can be applied to the primitive.

The condition for a modifier has the following format:

@Data("Group_Name.Control_Object_Name.Variable") [comparator] VALUE

for example:

@Data("FCX01.Pump1.Pv") == 2

The comparator available are:

\[<, >, <=, >=, =, <>\]

equal to
not equal to
less than
less than or equal to
greater than
greater than or equal to
3.4.6 Linked Parts

A group of primitives can be grouped together and saved to be re-used. This is known as a **Linked Part**. In addition, there is a library of existing linked parts that are available for putting on a graphic. A linked part contains a **Data Bind** definition, in which object names, etc need to be linked to **generic names**.

Available Linked Parts include:

- PAS_POU – these link directly to PAS_POU blocks in the FCX
- PROCESS_TAG – Similar to PAS_POU but link to other controllers
- ISA – Standard ISA symbols
- Motor
- Others – Conveyor, Fire, Stack
- Pipe
- Pump
- Switch
- Tank
- Valve
To place a linked part on a graphic, select the required part and click and drag it on to the graphic. Once a part is on the graphic, the Data Bind must be set up.

- Right click on the object and select Properties.
- Click on the Data Bind tab.
- Check “Set an individual generic name”
- The following generic names must be defined (as a minimum):

<table>
<thead>
<tr>
<th>Generic</th>
<th>Binding (example)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SVRID</td>
<td>“”</td>
</tr>
<tr>
<td>$GROUP</td>
<td>“FCX01”</td>
</tr>
<tr>
<td>$TAG</td>
<td>“FIC001”</td>
</tr>
<tr>
<td>$ITEM</td>
<td>“Pv”</td>
</tr>
</tbody>
</table>

$ITEM is not required for PAS_POU and PROCESS_TAG parts, but is required for all other parts.
3.4.7 HMI Deployment Tool

Once a graphic has been created it must be deployed from the Engineering Set into the Execution Set so that the Web Server can serve it as a graphic on the HMI clients. This is done using the HMI Deployment Tool.

When executed, the HMI Deployment Tool searches for graphics in the specified Engineering Set folder, and moves them into the Execution Set folder.

By default, the Engineering Set folder is:

\VDS\Work\HMI\HmiSet

Graphics are saved to the Graphics sub-folder of this folder.

A new Set can be created in which the Set Name, location of the Engineering Set, and other parameters are changed.

The options available with the Deployment Tool are:

- **Overwrite**
  - New Files Only (default)
  - Always

  If ‘New Files Only’ is selected, only files that have been changed will be deployed.

- **File Type**
  - All
  - Graphic (default)
  - HTML

  If ‘Graphic’ is selected, then only graphic type files will be deployed.
3.5 Historian and Trend Function

The historian function collects data and stores them in the VDS historical database. This data can then be used in the trend display function on the graphics, or accessed through OPC or the VBA Engine.

The History Builder is accessed through the Object Builder.

**Capacity**
- 32 blocks
- 4 groups per block at 8 pens per group
- maximum of 4 blocks can be 10 second scan or faster

### 3.5.1 Structure of the Historian

- **Blocks** – a block is a data acquisition unit with a set scan, type and other attributes.

- **Groups** – each block has 4 groups into which a set of Control Object variables are defined. Up to 8 variables can be defined for each group.
3.5.2 Function of the Historian

The data specified within the trend groups are collected at the rate defined for the block. The data is held within the block for the period specified by the Acquisition Period. After that it is either deleted, or moved to the Long Term History file, where it is held for a specified period.

The setup parameters for a History Block are:

- **Block Name**
- **Sampling Period** – the rate at which the data is sampled
- **Acquisition Period** – the period for which the data is held
- **Acquisition Method** – whether the block is continuous or batch sampling (see below)
- **Trigger Tagname** – if the trend is batch type, this variable must be true to trigger sampling
- **Dead band Limit** – the percentage change in input required for the value to be collected
- **Delay**
- **Pretrigger acquisition time period** – for batch trend types, this defines the scan period for the trigger.
- **Long-term Storage** – Specify whether long term storage is enabled.
- **Storage filename** – name and location of the long term storage file for this block.
Acquisition Method

The block acquisition method types are as follows:

- **Continuous/Rotary Trend** – continuous sampling of data
- **Batch/Rotary Trend** – a trend that can be started and stopped, but behaves like a continuous rotary trend when running. Triggered by the variable specified in the Trigger Tagname.
- **Batch/Non-Rotary Trend** – a trend that can be started and stopped, but stops when the trend block is full. Triggered by the variable specified in the Trigger Tagname.
- **Non-acquiring** – does not collect data. Used for such situations where an external application gathers data and displays the data using the VDS function.
3.5.3 Procedures

- **Procedures for setting up the Historian:**

1. In the Object Builder, click on Tools → History Builder. The History Builder window appears with a list of History Blocks that have been created.

2. To create a new History Block, select PROJECT → Add History Block. The History Block Details window appears:

3. Enter the configuration information for the block in the GENERAL tab.
4. Select a Group to enter trend points to be collected.
5. To add a point, double-click on a Group name in the Object Window, and select the required Control Object (tag).
6. Variables that belong to the data structure for that tag appears in the Properties box. Select the required variable and click on the ADD button.
7. This process can be repeated for up to 8 pens per group.
8. Click OK when finished.
3.6 Report Function

3.6.1 General

The Report Function is used to collect, process and print process data on a pre-defined basis.

The Report Function carries out the following functions:

- **Collects Data** at a given acquisition rate
- **Calculates** averages, totals, and other ‘closing’ data and stores the values to disk at a given recording interval
- **Prints** the report at a specified interval, either to disk as an Excel spreadsheet, or to printer
3.6.2 Report Functions

The Report Function has the following report types:

- **Daily** – Continuous daily printing of the report
- **Monthly** – Continuous monthly printing of the report
- **Daily/Monthly** – Combined daily and monthly printing of the report
- **Lot** – Event driven report that collects data while a condition is true
- **Batch** – Event driven report that samples data when an event becomes true

The closing process is a calculation function within the report function that calculates summary data from the raw data that is read in. The calculations that are available are:

- Average
- Total
- Maximum/Minimum
- Time at Maximum/ Minimum
- Difference Total
- Acquisition Status

Calculations can be made in one of two modes:

- **Recording Mode** – the calculation is made at the time of acquiring the data
- **Closing Mode** – the calculation is made at the ‘closing’ time (usually once a day when the report is printed)
3.6.3 Data Collection and Processing Function

The data is collected at a given rate (Acquisition Interval) and written to the ISAM historical database at the Recording Interval. Calculations can be performed at the Recording time as well as the Closing (i.e. Printing) time.

If there are no calculations done during the Recording process (i.e. Instantaneous Value), then only the latest Acquisition Value is recorded. In that case, the acquisition and recording intervals should be the same.

**Example 1** – if the Recording process is defined as Instantaneous Value (i.e. no calculation), and the Closing Process is defined as Average, then the average of all of the Recorded Instantaneous values is calculated at the time of printing.

**Example 2** – if the Recording process is defined as Average, then the data that is stored to disk at the Recording interval is the average of the Acquired data. If the Closing Process for that point is also Average, then the average of all the averages over the closing period is calculated at the time of printing.

Note that one data point can be assigned to several Closing processing calculations (modes), but only one calculation can be assigned to it during the Recording process.
3.6.4 Batch Reporting Function

If the report is a ‘Batch’ or ‘Lot’ type, then recording and batch start/end conditions must be defined.

- For a Batch type report, both the recording process can be started and stopped due to an event, and the report creation process can be started and stopped due to an event.
- For a Lot type report, only the recording process can be started and stopped. When the recording process is stopped, then a report is automatically generated.

The Recording process options are:

- **Polling** – the recording process starts and stops due to a polling event condition
- **Event-Driven** – the recording process is driven by a command in the FCX
- **Everyday** (Batch reports only) – the recording process is continuous, with daily closing processing

The Batching process options are (Batch type reports):

- **Polling** – the closing and printing process is carried out at the change of a polling event condition
- **Event-Driven** – the closing and printing process is driven be a command from the FCX

In the Polling Condition tab, the Recording or Batching process is selected and the Control Object is assigned to the process. The Start condition causes the process to start if it is True and stop if it is False.
3.6.5 Selecting the Report Data

The Acquisition Data tab selects the data to be recorded for the report. The Control Object and its Variable (Property) is selected and a Field Name is assigned to it. Click the ADD button to assign it to the list of collection data.

Each point is assigned a Recording Mode. See section 3.5.3 for more information on Recording mode functions.
3.6.6 Closing Processing Configuration

For each point that is assigned in the Acquisition Data tab, several Closing Modes can also be assigned to it.

Procedure:

1. select a Closing Mode
2. select the required data by clicking in the SET field
3. click on the ADD button
4. this procedure can be repeated as many times as required so that each data point can be assigned to several Closing modes.
3.6.7 Report Function Setup

The Report Function Setup utility is accessed by selecting PROJECT ➔ PROJECT SETUP.

![Project Setup](image)

The following items can be specified:

- Report file location
- Database Setting
  - **Computer Name** – name of computer in which the database is located
  - **Service Name** – ISAM data server service name
  - **User name** – ISAM data server login name
- Manual print/indication
3.7 Message Function

The message display on the graphics must be first setup to capture and process the process and system messages. The schematic below shows the function of the Message Management Server.
The message function must be configured at the FCX level and at the VDS level.

**FCX Message Configuration:**

In the Object Builder, select the FCX. One of the objects in the right-hand window is the FCX. Double-click on it to bring up the Properties window:

**History Catch-up of Messages:**

During VDS start-up, it gathers messages stored inside the FCX back to the specified Reread time. It does not gather data prior to this time, even if the data exists. In this case, it outputs a message to the Windows event log advising that the old messages were not gathered.
VDS Message Configuration:

VDS Message Configuration sets-up all parameters associated with the recording and presentation of alarms.
3.8 Security Function

The security builder defines the access levels of Operators using the operator display functions. See section 3.2.2 for information on setting up users. Users who are operators will be members of a VDS Operator’s group. These are groups defined in Windows security. There are 7 VDS Operator groups.

Security configuration is accessed through the Object Builder in the TOOLS menu.

Security setup is defined in the following categories:

- **General** – general setup for all users
  - Enable/Disable Security function
  - Setup timeouts for all users

- **User Group** – individual users security access configuration
  - *Data Server group access control* – Setup access privileges for each FCX and other controllers
  - *Other access control*
    - Historical Data Acquisition access level (2 levels)
    - Message Management access level (2 levels)
  - *Function Privilege*
    - System Programs
    - Tuning Panel
    - Mode Switch
    - Security Log

- **Security Log** – for each of the above mentioned categories, setup read/write access for the log files.
3.9 HMI Client/Server Functions

3.9.1 HMI Server Configuration

The HMI Server reads data from the Data Server and makes it available to the HMI Clients. If the HMI Server is in the same PC as the Data Server, then no configuration is required. However, if it is in a different PC, then the location of the Data Server needs to be specified in the HMI Configurator.

![HMI Configurator](image)

The HMI Server can connect to multiple Data Servers, and these are specified as follows:

- **Server ID** – Data Server ID service name
- **Server Type** – Always VDS
- **Default** – Select which Data Server is the default server for this Web Server
- **Computer Name** – The name of the PC in which the Data Server resides
3.9.2 Operator Display Function

Operator displays are displayed on Internet Explorer web pages in a Web Client.

The following types of displays are available:

- Logon/Logoff Windows – for user security
- Graphics Windows – display graphics
- Objects Windows – display Control Objects as faceplates

When running the web client, use the following address to connect to the HMI Server:

http://localhost/Stardom/Hmi/Run/HmiSet/index.html

where ‘localhost’ is the name (or IP address) of the Web Server. If the client is running in the same PC as the server, then use the word ‘localhost’.

![Web Client Screenshot](image-url)
Before displays can be called up, the user must logon to the server. This is done by clicking on the LOGON link to display the Logon window:

The default is for no User ID or Password.

The user can hide and display certain frames within the web page. These are:

- **Index** – the left hand frame with the list of graphics and objects
- **Resident** – the frame just above the graphic display frame
- **Object View** – the left-hand frame for displaying faceplates

These frames can be hidden by clicking on the ‘X’ button.
Displaying a Graphic

To call up a graphic, first click on the ‘graphics’ link to display the graphic index. The list of graphics is displayed in the frame below.

There are several options when displaying a graphic:

1. Clicking on the Graphic name displays a full sized graphic in the adjacent graphic frame.
2. Clicking on the ‘*’ to the left of the graphic name displays a full sized graphic in a new window.
3. Clicking on the ‘Half’ displays a half sized graphic in the adjacent graphic frame.
4. Clicking on the ‘*’ to the left of the ‘Half’ displays a half sized graphic in a new window.

Displaying an Object

Clicking on ‘Object’ displays the Object Index in the Object frame.

Clicking on an object name will display the faceplate for that Control Object.

Note that there are the same display options available as there are for calling graphics.

A search facility is available to search for tag names. This is necessary if there are thousands of objects in the list.
SECTION 4

TUTORIALS

VDS Versatile Data Server Software

Application Portfolio

Autonomous Controllers
CONTENTS

4.1 Introduction ........................................................................................................................................... 4

4.2 Resource Configurator .............................................................................................................................. 5
    ❑ Procedure 1 - FCX IP Address Setting ................................................................................................. 5
    ❑ Procedure 2 - Display Mode .................................................................................................................. 6
    ❑ Procedure 3 - License Registration ....................................................................................................... 7
    ❑ Procedure 4 - To Create a Resource ...................................................................................................... 8

4.3 Logic Designer ........................................................................................................................................ 9
    ❑ Procedure 1 - Starting Logic Designer: .................................................................................................. 9
    ❑ Procedure 2 - Create a new project: ....................................................................................................... 9
    ❑ Procedure 3 - Add a library: .................................................................................................................. 10
    ❑ Procedure 4 - Create a new Program POU: ......................................................................................... 11
    ❑ Procedure 5 - Create a Configuration .................................................................................................. 12
    ❑ Procedure 6 - To Create a Resource .................................................................................................... 13
    ❑ Procedure 7 - Setting the FCX Address .............................................................................................. 13
    ❑ Procedure 8 - To Create a Task ............................................................................................................. 14
    ❑ Procedure 9 - To Assign a Program to a Task .................................................................................... 16

4.4 Creating an IEC61131 Program ............................................................................................................. 17
    ❑ Procedure 1 – Create Function Blocks .............................................................................................. 18
    ❑ Procedure 2 – Create and Connect Variables ................................................................................... 19
    ❑ Procedure 3 – Connect the Function Blocks ..................................................................................... 20
    ❑ Procedure 4 – Make and Download the Project ................................................................................. 21
    ❑ Procedure 5 – Running the Program .................................................................................................. 23
    ❑ Special Notes ....................................................................................................................................... 24

4.5 Creating a PAS POU Program ............................................................................................................. 25
    ❑ Procedure 1 – General Setup: ............................................................................................................. 26
    ❑ Procedure 2 – Define the I/O configuration: ....................................................................................... 26
    ❑ Procedure 3 – Create the PID Controller ........................................................................................... 28
    ❑ Procedure 4 – Create the Input/Output Function Blocks ..................................................................... 29
    ❑ Procedure 5 – Connect Readback Variables ....................................................................................... 30
    ❑ Procedure 6 – Connect I/O Variables ................................................................................................. 31
    ❑ Procedure 7 – Engineering Parameter Connection ............................................................................... 32
    ❑ Procedure 8 – Softwiring ...................................................................................................................... 33
    ❑ Procedure 9 – Make and Download ..................................................................................................... 34

4.6 VDS Display of PASPOU Program ...................................................................................................... 35
    ❑ Procedure 1 – Object Builder .............................................................................................................. 35
    ❑ Procedure 2 – IP Address Setting ....................................................................................................... 36
    ❑ Procedure 3 – Mode Change to Debugging ......................................................................................... 37
    ❑ Procedure 4 – Tuning Panel ................................................................................................................ 38
    ❑ Procedure 5 – Browser Display .......................................................................................................... 39
    ❑ Procedure 6 – RAS Panel ..................................................................................................................... 40

4.7 Extension to PAS POU Program ......................................................................................................... 41

4.8 Graphics Builder .................................................................................................................................. 42
    ❑ Procedure 1 – Preparation .................................................................................................................. 43
    ❑ Procedure 2 – Create Graphic ............................................................................................................ 45
Procedure 3 – Draw a square ................................................................. 45
Procedure 4 – Create Process Data Character........................................... 45
Procedure 5 – Graphic Modifier .............................................................. 46
Procedure 6 – Define the Faceplates ......................................................... 47
Procedure 7 – Create pushbuttons for calling up graphics ....................... 48
Procedure 8 – Defining the History Builder ............................................. 49
Procedure 9 – Create Trend Graphic ....................................................... 50
Procedure 10 – Create the Message Window .......................................... 51
Procedure 11 – Create the History Window ............................................ 51
Procedure 12 – Deploy the Graphics ....................................................... 52
Procedure 13 – Review Operating Graphics ............................................ 52
4.1 Introduction

This section covers a tutorial that takes the student through the various steps described in Sections 1 to 3 for system generation.

It is divided into sections in accordance with the previous sections of the manual so that they can be done as the various sections are covered, or may be done as a single exercise.

As each one follows on from the other, it is important that the tutorials are done in the correct order.
4.2 Resource Configurator

- **Procedure 1 - FCX IP Address Setting**

1. **Cabling**
   Connect an Ethernet cable to the network interface **port 1** of the FCX.

2. **Start “Resource Configurator”**
   Start from a start menu [start]-[programs]-[stardom]-[YOKOGAWA FCN-FCJ]-[Resource Configurator]

3. **Power on / Reset**
   Power on or push the reset switch of FCX.
   After the LED starts blinking, push the shutdown button once within 3 Sec.

4. Make sure that “message area” is visible (if not click VIEW – Message Area). The MAC address will appear on the Resource Configurator

5. Click on FILE – IP Address Setup and type in the required IP address of the FCX. The recommended address format is: **xxx.yyy.0.1** (e.g. 176.16.0.1).

Note, the FCX can be identified by the MAC address on the side of the CPU module for FCN, and on the back of the unit for FCJ.

If the FCX has a dual-redundant CPU configuration, then the second CPU IP address has to be set using the APC function in the Tools menu. APC means All Program Copy and copies the contents of the first CPU to the second. This also causes the CPU address to be set.

6. Set the PC IP address. In the TCP/IP address setup for the PC, two IP addresses need to be set, e.g. 176.16.0.100 and 176.16.1.100 (Virtual and Physical).
Procedure 2 - Display Mode

Display Mode Change
- Tree
- Graphic
Procedure 3 - License Registration

1. Click on the License Tab.
2. Click on “Taking in of License File”.
3. Select the files from the floppy disk.
4. Select FILE – DOWNLOAD to download the licenses to the FCX.
5. Restart FCX (Power OFF \(\Rightarrow\) ON)
Procedure 4 - Backup Settings – Maintenance Web Page

Upload the Backup Tool from the FCX:

1. Make sure that the web browser does not have proxy server selected.
2. Connect to the following address: http://<IP address or hostname>/MNT.
3. Specify **user name**: “stardom” and **password**: “YOKOGAWA”.
4. Choose “Maintenance Menu”.
5. Choose “Download the Backup Tools” and download the compressed file.
6. Return to “Top Page”.
7. Unzip the file.

Using the Backup Tool:

From DOS prompt:

- **Backup**
  - FcxBackup <host name or IP address>

- **Restoration**
  - FcxRestore <host name or IP address>
4.3 Logic Designer

- **Procedure 1 - Starting Logic Designer:**

  Click on [Start]-[Programs]-[stardom]-[YOKOGAWA FCN-FCJ]-[Logic Designer]

  The Logic Designer starts with the previous project loaded.

- **Procedure 2 - Create a new project:**

  1. Select FILE - NEW PROJECT
  2. The New Project Template selector appears:

     ![New Project Template](image)

     3. Select **STARDOM PAS**. Selecting either of the other two creates a project without the Yokogawa Function Block libraries, which would have to be added manually.

     4. To give the project a name, select FILE – SAVE PROJECT AS, and type in the required project name (EX01A).

     Note that when the project is created, it creates a complete Configuration, with a Program (Main), an FCX (FCX01) a task (Task1 - Cyclic) with the program instantiated in.
Procedure 3 - Add a library:

1. Right click on ‘Library’ to insert library, or select ‘Library’ and click on the PROJECT menu and select ADD LIBRARY.

2. Select the required library (.mwt file) and click OK.

Because STARDOM PAS was selected for new project, all of the above libraries should already exist in the project.
Procedure 4 - Create a new Program POU:

1. Right click on ‘Logical POU’, or select it and click on the PROJECT menu.
2. Select “Function Block”

3. The following dialog appears:

4. Type in the name: **FBD001** and select **PROGRAM**.

5. It is recommended that the **PLC type** and **Processor type** be selected as **<independent>** to ensure portability of the configuration.

6. These values can all be changed at any time by right clicking on the POU and selecting **PROPERTIES**.
Procedure 5 - Create a Configuration

Note that because a configuration is automatically created when the project is created, this step is not necessary. This is provided for information.

1. Right click on Physical Hardware and select INSERT - CONFIGURATION

2. Enter the Configuration name (PLC type is always IPC_32).
Procedure 6 - To Create a Resource

Note that because FCX01 is automatically created when the project is created, this step is not necessary. This is provided for information.

1. Right click on a Configuration and select INSERT - RESOURCE

![Configuration Resource Insert](image)

2. Enter the Resource name – FCX01 (Processor type is always FCX).

![Resource Insert](image)

These parameters can be changed after the FCX is created by right clicking on the resource and selecting PROPERTIES.

Other parameters for the FCX can be set by right clicking on the resource and selecting SETTINGS.

Procedure 7 - Setting the FCX Address

Double click on Target Setting and type in the IP address of the FCX as follows:

![Target FCX Address](image)
Procedure 8 - To Create a Task

1. Right click on a Resource (FCX) and select INSERT - TASK

![Diagram of Stardom interface showing task creation process]

2. Enter the Task name and Task type.

![Insert window showing task parameters]

These parameters can be changed after the task is created by right clicking on the task and selecting PROPERTIES.
3. Enter task priority and scan rate through the next dialog that appears:

![Task settings for IPC_32 dialog box]

These parameters can be changed after the task is created by right clicking on the task and selecting SETTINGS.
Procedure 9 - To Assign a Program to a Task

To instantiate a program, the program must be assigned to a task.

1. Right-click on Task1
2. Select Insert → Program Instance

3. Select FBD001 for Program Type
   
   note: All POUs designated as Programs will be in the drop-down list for Program Type.

4. Type in Program Instance name (FBD001)
   
   note: Any name can be used for the Program Instance. It does not have to be the same name as the Program Type.
4.4 Creating an IEC61131 Program

The aim of this tutorial is to create a simple program using standard IEC61131 Functions and Function Blocks. It is necessary to do Tutorial 4.3 before this one, because it sets up the project and program to carry out this exercise.

The program appears as follows:

If Inputs A001 and B001 are both true, then a timer will start counting up. When 15 seconds have expired, the output, C001, turns on.
Procedure 1 – Create Function Blocks

1. Double click on FBD001 in the Project Tree:

A blank worksheet appears.

2. Open the EDIT WIZARD in the VIEW menu. This contains all of the functions and function blocks organized by library. Select the FUNCTION library:

3. Click on the place in the worksheet where the function is to be positioned, and double click on the AND function. The AND function will appear on the worksheet.

4. Click on the place in the worksheet where the Timer Function Block is to be positioned.

5. Select Function Block in the Edit Wizard and double-click on the TON block. The TON will appear on the worksheet.

6. Because the TON is a Function Block, it requires a Label Name. Type in TON001.
Procedure 2 – Create and Connect Variables

7. Create variables and connect them to the AND block by double-clicking on the two inputs and entering the variables A001 and B001. These are local variables of data type BOOL:

8. The variables can be checked by opening the FBD001V Variables File:
9. In the same manner, define the variable for the TON block:

![Diagram of TON block]

- TIME#15s: literal (do not set a data type)
- C001: BOOL
- ET001: TIME

- Procedure 3 – Connect the Function Blocks

1. Connect the AND and the TON together:

![Diagram of connection procedure]
Procedure 4 – Make and Download the Project

1. **MAKE** the program. Because it has already been instantiated into a task in Tutorial 4.3, it is not necessary to do it here. **MAKE** can be initiated by pressing F9, or through the BUILD menu, or by clicking on the MAKE icon.

   ![MAKE icon](image)

2. Check the message window. If there are errors, click on the ERRORS tab and double click on the first error. It will open the worksheet in which the error exists, and highlight the connection or function where the error has occurred.

   ![Message window](image)
3. Download the program. Open the PROJECT CONTROL dialog through the ONLINE menu, or by clicking on the PROJECT CONTROL icon.

From the PROJECT CONTROL dialog, select **Download**. The Download dialog appears. Click on **Download** in the **Project** section.

Note: downloading to Project will cause the FCX program to stop. Downloading to Bootproject will not, but power needs to be cycled to the FCX for this program to become active.
Procedure 5 – Running the Program

1. Run the Program. Once download is complete, the program needs to be started. Click on COLD from the PROJECT CONTROL dialog to start the program. Note that if a Cold start is initiated, there is a 60 second wind-up period before the program starts functioning.

2. Start Debugging Mode. Select the ONLINE menu, or click on the Debug icon, press F10.

The worksheet displays the values of the inputs and outputs. Inputs are set by double clicking on the input and setting the value in the setting dialog box.
The program can be created using other methods of programming available through IEC61131. Examples using Structured Text and Ladder Diagrams are shown below.

**Ladder Diagram:**

![Ladder Diagram](image)

Note that this Ladder Diagram includes the TON Function Block. In other words, the different program types can be mixed on the one worksheet.

**Structured Text:**

```
TON001(IN:=(A001 AND B001),PT:=TIME#15s);
C001:=TON001.Q;
ET001:=TON001.ET;
```

Note that it is not necessary to write the program out directly. Just as with Function Block Diagrams, the functions and blocks can be selected through the EDIT WIZARD.

⇒ More information on IEC61131-3 programming is covered in the IEC61131 Fundamentals Training Course.

⇒ The worked solution to this tutorial is in the EX01 project in the PROJECTS folder of the training CD.
4.5 Creating a PAS POU Program

This tutorial creates a single loop control program using the Yokogawa PAS POU function blocks. It also introduces the procedures for I/O configuration, and simulation using softwiring.

Note that this tutorial assumes that tutorials 4.3 and 4.4 have been completed and that a detailed explanation of the procedure covered in those tutorials is not necessary.

The control loop to be created is as follows:

The final function block design in Logic Designer is as follows:
Procedure 1 – General Setup:

1. Create a new project and select STARDOM PAS.
2. Save it as LoopControl
3. In the Target Setting of the FCX, set the IP address.

Procedure 2 – Define the I/O configuration:

1. Device Label Configuration:

Create the following:

- Analog Input: AI001
- Analog Output: AO001

In the device label definition, as well as device label, the High and Low scales of the I/O (real numbers), and the Engineering Unit (text), can be configured.

Confirm that the I/O have been assigned as Global Variable by double-clicking on the Global Variable file:

Note that there are two Analog Output Global Variables created: AO001 and AO001_RB. AO001_RB is the read-back value that is used in the control loop as described below.
2. I/O Definition in the Resource Configurator

Run Resource Configurator and connect to the FCX.

Go to the analog module and type in the Device Label Names for the I/O points. These can be typed in directly, or can be copied and pasted from the Device Label Definition file in Logic Designer.

Download the changed configuration by selecting Download in the FILE menu.
Procedure 3 – Create the PID Controller.

1. Double-click on “Main” in the Project Tree. The worksheet appears. Delete any functions or blocks that are on the worksheet.

2. Open Edit Wizard from the VIEW menu. Select SD_PASPOU_PF. This is the library containing the Yokogawa control function blocks.

3. Click on the place in the worksheet where the function block is to be placed and double click on the PAS_PID function block in the Edit Wizard.

4. The Variable Properties dialog appears. Type in the name of the block, FIC001, and click OK. The PAS_PID block appears on the screen with the name FIC001.

5. Double-click on the PRM_WT terminal and type in “WPRM_FIC001”. Set the data type to STRING_PRM.

Note that the data type can be determined by right-clicking on the PRM_WT and selecting Properties. The data type of the terminal is confirmed as STRING_PRM.
Procedure 4 – Create the Input/Output Function Blocks

1. Place a PAS_AI_ANLG block from the SD_PASPOU_PF library on the left side of the PID block. Use the default name for the block (i.e., PAS_AI_ANLG_1).

2. Place a PAS_AO_ANLG block from the SD_PASPOU_PF library on the right side of the PID block. Use the default name for the block (i.e., PAS_AO_ANLG_1).

3. Connect the OUT of the PAS_AI_ANLG to the IN of the PID block.

4. Connect the OUT of the PID block to the IN of the PAS_AO_ANLG block.
Procedure 5 – Connect Readback Variables

- Create a readback variable to connect the readback output of the output block to the readback input of the PID block.

1. Double-click on RB_IN of the PID block. The Variable Properties dialog appears.
2. Type in the name FIC001_RB
3. Set data type to CData_REAL. Click OK.
4. The variable FIC001_RB will connect to the RB-IN terminal of the PID block.
5. Double-click on the RB_OUT block of the Output block and select FIC001_RB. Click OK.
Procedure 6 – Connect I/O Variables

1. Double-click on the IN terminal of the PAS_AI_ANLG block.

2. In the Variable Properties dialog, click on the Global Scope tab and select DeviceLabel_Input_A.

3. Click on the Variables tab, select Global, and select AI001. Click OK

4. Double-click on the OUT terminal of the PAS_AO_ANLG block. In the Variable Properties dialog, click on the Global Scope tab and select DeviceLabel_Output_A. Click on the Variables tab, select Global and select AO001. Click OK

5. Double-click on the RB_IN terminal of the PAS_AO_ANLG block. In the Variable Properties dialog, select Global and select AO001_RB and click OK.
Procedure 7 – Engineering Parameter Connection

To access and set engineering parameters in the PID block, the best way is to copy and paste a pre-defined ST function block from the PAS_EXAMPLE project (that comes with Logic Designer) called EPRM_PID. This block contains all of the engineering parameters in a PID block and the initial values can be edited directly.

1. Open another Logic Designer application.
2. Open the PAS_EXAMPLE project. [FILE] – [OPEN PROJECT]
3. Select EPRM_PID in the Project Tree and COPY. Close this Logic Designer (this can be done later). Go back to the original Logic Designer window.
4. Click on Logical POUs in the Project Tree and PASTE. The EPRM_PID function block will appear in the Project Tree.
5. Right-click on EPRM_PID, select Properties and change its name to EPRM_FIC001.
6. On the Main worksheet, in EDIT WIZARD, select the group with the same name as the project (LoopControl). A list of functions and function blocks created in this project will appear.
7. Click on the worksheet to the lower left of the PID block and double-click on EPRM_FIC001 in the EDIT WIZARD. The block will appear on the worksheet. Name it: EPRM_FIC001_1 (the default name).
8. Connect the ENGPRM of the EPRM_FIC001 block to the ENG_PRM terminal of the PID block.
9. Double-click on the ‘CMT’ terminal of the EPRM_FIC001_1 block and type in the comment ‘FIC001’. This will then be the comment for the faceplate. Note that any text could be written here.
10. Note that there are other parameters that can be set, and further inputs can be added to the block by configuring the appropriate variables as Input Variables.
Procedure 8 – Softwiring

Softwiring allows for the looping back of I/O connections where real I/O is not available.

1. Double-click on Softwiring and type in the following information:

   Wiring Destination 1       – AI001
   Wiring Source 1        – AO001
   Wiring Logic              – SD_WIRE_DTANLG_ANLG
   First-Order Lag Time     – t#5s

   Click OK. A POU is created called: FCX01_WIRE.

2. Instantiate this POU in Task1 of FCX01.

3. Open the Global Variables file and set the initial value of GS_NFIO_DISCONN to TRUE. This will disconnect the I/O from the software.
Procedure 9 – Make and Download

1. Select MAKE from the BUILD menu, or click on the MAKE icon.

2. Download the project. See the procedure in Tutorial 4.4, Procedure 13 for more information.

3. Perform a Cold Start.

4. The program should now be downloaded and operational.
4.6 VDS Display of PASPOU Program

This tutorial is a continuation of Tutorial 4.5. The purpose of this tutorial is to demonstrate how to configure the VDS to view the project loaded in the FCX.

- **Procedure 1 – Object Builder**

  1. Start the Object Builder:

  2. Select a new configuration:
     - [FILE] – [NEW]

  3. Import the taglist from the LoopControl project:
     - Select [File] – [Import Settings]
     - Click on Selection and click OK
     - Check “Use the last import information”
     - Select “Stardom ADLST” and then OK

     The file location is:
     C:\YOKOGAWA\FCN-FCJ\LogicDesigner\Projects\LoopControl\C\Configuration\R\FCX01

     - Select ADLST.csv
     - Click OK
     - Check “Update import settings”
     - Click OK
Procedure 2 – IP Address Setting

1. Select IO Object in the Object Builder.
2. Double-click on FCX01.
3. Click on the OTHERS tab.
4. In Resource Name, set the IP address of the FCX. IF the FCX name and address is set up in the LMHOSTS file in windows, then the FCX name (FCX01) can be used instead.
Procedure 3 – Mode Change to Debugging

1. Save the Project.

2. Click on the Mode change icon, or select Mode Change from the Tools menu.

3. Click on debugging and click OK.

4. Visual Basic will start, and this can be exited as it is not required for this exercise.
Procedure 4 – Tuning Panel

1. In Object Builder, click on I/O Object.

2. Select the Tuning Panel - [start]-[program]-[stardom] - [Support tool] - [FCJ FCN tuning panel]

3. Double-click “Main1FIC001” of FCX01 group.
Procedure 5 – Browser Display

The operator display is a web browser page that accesses data through the HMI Server of the VDS.

1. Start Internet Explorer

2. Type in the address: http://localhost/Stardom/Hmi/Run/HmiSet/index.html

3. This should be book-marked, and in the case of the operator station, set as the home page.

The following page will appear:

4. Click on “Logon” and click OK (i.e. no user name or password)

5. At this stage there are no graphics, but in the Objects frame, it is possible to call up the tag FIC001 and view it as a faceplate.

6. It is also possible to call up the RAS panel (see next procedure).
Procedure 6 – RAS Panel

RAS – Reliability Accessibility Serviceability

This panel can be called up as a standard display. It does not have to be created by the user. It provides status information about the FCX, such as the status of the CPU card, power supply and I/O modules.

To call it up in the web browser:

- click on Objects
- FCX01.FCX01

The RAS panel for the FCX will appear.
4.7 Extension to PAS POU Program

In this section, we extend the PAS POU exercise further, using a Program Set block to drive the setpoint of the PID block. This is required for the graphic exercise in section 4.8.

The procedure is as follows:

1. Open the LoopControl exercise in Logic Designer.

2. Select a PAS_PG_L30 block from the PAS POU library and place it on the worksheet. Configure the name as “FSC001”.

3. Double-click on the PRM_LN terminal and configure the name as “WPRM_FSC001”. The variable type is “STRING_PLN”.

4. Double-click on the RB_IN terminal and configure the name as “FIC001_RB”. The variable type is “CDataReal”.

5. Open another Logic Designer and open PAS_Example. Select EPRM_PG_L30 under LogicalPOUs and copy it. Go back to the original Logic Designer and paste it into the LogicalPOUs. Change its name to EPRM_FSC001.

6. In the Edit Wizard open the LoopControl category and place EPRM_FSC001 onto the worksheet. Name it EPRM_FSC001_1.

7. Connect the ENGPRM terminal of EPRM_FSC001_1 to the ENG_PRM terminal of FSC001.

8. Make and download the changes. Import the new ADLST into Object Folder, and run debugging mode.

9. In the tuning panel for FSC001, set PlotProgress (2) = 30
   PlotOutData (2) = 300
4.8 Graphics Builder

This exercise creates a graphic, trend, alarm summary and historical message, based on the PAS POU control exercise.

The graphic displays the faceplates of the PID block and the PG_L30 block, and a button to set the PID block into Cascade. It also has several buttons to call up the trend, etc windows.
Procedure 1 – Preparation

1. Set the mode in Object Builder to Design.

2. Export the data from the Object Builder:
   i. Click on [File] – [Export Setting]
   ii. Select “All” and click OK.
   iii. The Save As dialog appears with the directory set to [VDS] – [Work] – [DataServer]
   iv. Double-click on LoopControl and set the filename to LoopControl.CSV. Click OK.
   v. The Control Object data is exported to this file.

3. Run Graphics Builder
4. Create the Data Source:

The Data Source dialog is a convenient tool to allow drag-and-drop of tags into functions on the graphic.

i. In the Graphics Builder, click on [Tools] – [Generating DataSource]

ii. Specify HMI configuration file name – leave on default. Click NEXT.

iii. Data Server CSV File Name – select:

iv. C:\YOKOGAWA\VDS\Work\DataServer\LoopControl\LoopControl.csv

v. Click NEXT, then click OK twice.

vi. Click on [Tools] – [Datasource]

![Data Source Dialog Box]

<table>
<thead>
<tr>
<th>Data Source</th>
<th>Ser...</th>
<th>Type</th>
<th>Date of Cr...</th>
</tr>
</thead>
<tbody>
<tr>
<td>@Data(&quot;FCX01 FCX01 CpuCapacity&quot;)</td>
<td>Local</td>
<td>RAS</td>
<td>2002/03/03</td>
</tr>
<tr>
<td>@Data(&quot;FCX01 GS_RETAIN_RST_FG.Cv&quot;)</td>
<td>Local</td>
<td>DI</td>
<td>2002/03/03</td>
</tr>
<tr>
<td>@Data(&quot;FCX01 GS_RETAIN_RST_SW.Cv&quot;)</td>
<td>Local</td>
<td>AI</td>
<td>2002/03/03</td>
</tr>
<tr>
<td>@Data(&quot;FCX01 GS_RETAIN_RST_FG.Cv&quot;)</td>
<td>Local</td>
<td>DI</td>
<td>2002/03/03</td>
</tr>
<tr>
<td>@Data(&quot;FCX01 GS_RETAIN_SW.Cv&quot;)</td>
<td>Local</td>
<td>AI</td>
<td>2002/03/03</td>
</tr>
<tr>
<td>@Data(&quot;FCX01 Main1FSC001.Mv&quot;)</td>
<td>Local</td>
<td>PPID</td>
<td>2002/03/03</td>
</tr>
<tr>
<td>@Data(&quot;FCX01 Main1FSC001.Mv&quot;)</td>
<td>Local</td>
<td>PPICG</td>
<td>2002/03/03</td>
</tr>
</tbody>
</table>
Procedure 2 – Create Graphic


2. Select [File] – [Properties] and in the Attribute tab, set the graphic size to 980 x 615.

Procedure 3 – Draw a square

1. Click on the Square icon and draw a square on the left of the graphic page.

2. Right-click on the square and set the properties as follows:
   - **Line** – Transparent
   - **Fill** – Dark Blue

Procedure 4 – Create Process Data Character

1. On the page, click on the Process Data Character Button “0.0”.

2. Click in the square on the graphic so that the characters “RRRRRRR” appear.

3. Right-click on the characters and select properties. The following dialog appears.

4. Click on the “Text” tab and set the font size to 24, text colour to Cyan, and background colour to black.

5. Click on the “Process Data – Character” tab. In the Datasource dialog, click on the Main1FIC001.Pv tag and drag it into the Display Data field of the dialog.
6. Repeat this process for Main1FIC001.Sv and Main1FIC001.Mv. In this case, drag the Main1FIC001.Pv into the “Display Data” field and manually change the data item.

Procedure 5 – Graphic Modifier

Set up the blue square to go red if Main1FIC001.Mv > 10.0

1. Select the blue square, right-click and select Properties. Click on the Graphic Modify tab.

2. Click on Main1FIC001.Pv in the Data Source dialog and drag it into the “Conditional” field of the Properties dialog.

3. Modify the string to: @Data("FCX01.Main1FSC001.Mv") > 10.0


5. Click “ADD”. Click OK.
Procedure 6 – Define the Faceplates

The faceplates are linked parts. They are placed on the graphic and the generic data is defined through the data bind.

1. Click on the Linked Parts icon.
2. Go to LinkParts – Faceplate – PAS_POU, and select the PAS_PID faceplate.
3. Click on the faceplate in the adjacent window and drag it onto the graphic.
4. Right-click on the faceplate and select properties.
5. In the Data Bind tab, check “set an individual generic name”
6. Type in the following data, clicking ADD for each one:

<table>
<thead>
<tr>
<th>Generic</th>
<th>Binding</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SVRID</td>
<td></td>
</tr>
<tr>
<td>$GROUP</td>
<td>“FCX01”</td>
</tr>
<tr>
<td>STAG</td>
<td>“FIC001”</td>
</tr>
</tbody>
</table>

7. Repeat this procedure for tag FSC001.
Procedure 7 – Create pushbuttons for calling up graphics

Three pushbuttons are created to allow the calling up of the trend, alarm and historical message displays. These displays are covered in the next procedures.

1. Click on the Pushbutton icon.
2. Draw the pushbutton on the lower left-hand corner of the graphic.
3. Right-click on the pushbutton and select Properties.
4. In FUNCTION, set the following parameters:
   Function – Open monitoring or logon window
   Window – Graphic window
   Parameter - "window=Trend001"
5. Click OK.
6. From the toolbar, select the text tool “A”, and click on the pushbutton.
7. Type in the text: “TREND”.
8. Right click and select properties. Set the following parameters:
   Font size = 24
   Text color = Dark Blue
9. Click OK.
10. Select both the button and the text and copy and paste it to the right. Repeat so that there are three buttons.
11. Change the text on the second button to ALARMS, and the third button to HISTORY. Do this by selecting the text and clicking on the “A” icon to edit it.
12. Select the properties of each of these buttons and change the Parameter field as follows:
   Button 2 – “window = Message001”
   Button3 – “window = Hist001”
Procedure 8 – Defining the History Builder

Before the trend graphic can be created, the History needs to be defined in the History Builder.

1. In Object Builder, select History Builder from the Tools menu.

2. In the History Builder, select PROJECT → ADD HISTORY BLOCK.

3. In the General tab, set the following parameters:

   - Name: B001
   - Sampling Period: 2 sec
   - Acquisition Period: 4 hours
   - Acquisition Method: Continuous/Rotary Trend

4. Click on the Group 1 tab.

5. Double-click on FCX01 in the object window. The list of control object appears.

6. Click on Main1FIC001. The list of items appears in the Properties window.

7. Click on Pv, and click the ADD button.

8. Repeat for Sv and Mv.

9. Click OK. Click Save, and exit the History Builder. Click SAVE in the Object Builder.
Procedure 9 – Create Trend Graphic

1. In the Graphic Builder, select [FILE] – [Create New].

2. Go to [FILE] – [Properties] – [Attributes], and set the graphic size to 980 x 615.

3. Click on the trend icon and click on the top left-hand corner of the graphic. The trend will appear, taking up most of the graphic page.

4. Right click on the trend and select properties.

5. In Trend Data Source, type in: "B001.Group1"

6. Leave the trend size on small.

7. Save the graphic as “Trend001”.
Procedure 10 – Create the Message Window

1. In the Graphic Builder, select [FILE] – [Create New].

2. Go to [FILE] – [Properties] – [Attributes], and set the graphic size to 980 x 615.

3. Click on the message icon and click on the top left-hand corner of the graphic. The message primitive will appear, taking up most of the graphic page.

4. Right click on the message primitive and select properties.

5. For Primitive Type, select “Alarm Summary”.

6. ServerID needs to be configured only if there is more than one Data Server.

7. Save the graphic as “Message001”.

Procedure 11 – Create the History Window

Follow the identical procedure as for the Message Window, except that the Primitive Type is “Historical”.

Save the graphic as “Hist001”.

---

---
- **Procedure 12 – Deploy the Graphics**

  1. Save the main graphic as LoopGR.
  2. Open the HMI Deployment Tool. Leave on the default settings.
  3. Click EXECUTE.
  4. Confirm that all the graphics just created are loaded.
  5. Exit the Deployment Tool.

- **Procedure 13 – Review Operating Graphics**

  1. In Object Builder, select Debugging mode. Close the Visual Basic window.
  2. Run Internet Explorer.
  3. Set the following URL: [http://localhost/Stardom/Hmi/Run/HmiSet/index.html](http://localhost/Stardom/Hmi/Run/HmiSet/index.html) (this should already be book-marked).
  4. Logon to the server (no user name or password).
  5. Click on “Graphics” and select LoopGR.
  6. Confirm that the faceplates and display data are reading the correct data.
  7. With the PID block set to MANUAL, set the MV above 10% and confirm that the blue square goes red.
  8. Click on the trend button and confirm that the trend window appears, and that the correct data is being collected and displayed.
  9. Create a HI alarm in the PID block and open the ALARMS page. Confirm that the alarms appear in the window.
  10. Open the HISTORY page, and confirm that alarm and other messages appear on this display.