# Function Block Programming for Distributed Control

A thesis presented in complete fulfilment of the requirements for the Master in Engineering

216.899 Thesis

at Massey University, Wellington
New Zealand.

**Andrew Robert Meek**

**July 2004**

I declare that this thesis is my own, unaided work. It is being submitted in complete fulfilment of the requirements for the Master of Engineering at Massey University. It has not been submitted before for any degree or examination in any other University.

.....

15th day of August, 2005

# MASSEY UNIVERSITY
## APPLICATION FOR APPROVAL OF REQUEST TO EMBARGO A THESIS
## (Pursuant to AC 98/168 (Revised 2), Approved by Academic Board 16.02.99)

Name of Candidate: Andrew Meek ID Number: 00213918

Degree: Masters Engineering Dept/Institute/School: IIS&T

Thesis Title: Function Block Programming for Distributed Control

Name of Chief Supervisor: Frans Weehuizen Telephone Extn: 6798

As author of the above named thesis, I request that my thesis be embargoed from public access until (date) 1st July 2007 for the following reasons:

- [x] Thesis contains commercially sensitive information.
- [ ] Thesis contains information which is personal or private and/or which was given on the basis that it not be disclosed.
- [ ] Immediate disclosure of thesis contents would not allow the author a reasonable opportunity to publish all or part of the thesis.
- [ ] Other (specify): _____

Please explain here why you think this request is justified:

Thesis contains commercially sensitive information and clause 6 of the project agreement signed by the student, Tait Control Systems and Massey University states: "As the project has potential commercial value, the thesis shall be embargoed for a period of 3 years." Failure to embargo this thesis would break the terms and conditions of the project agreement.

Signed (Candidate): _____ Date: 24/5/04

Endorsed (Chief Supervisor): _____ Date: 9-6-04

Approved/~~Not Approved~~ (Representative of VC): _____ Date: 11-6-2004

*Note: Copies of this form, once approved by the representative of the Vice-Chancellor, must be bound into every copy of the thesis.*

11 JUN 2004

# Abstract

This report discusses research and development using the draft IEC 61499 function block standard for distributed control with embedded microprocessor applications. This is a function block programming language that is currently under development for programming distributed control systems. The report covers what is required to develop an IEC 61499 compliant product and its suitablity for use with distributed control systems. To utilise the IEC 61499 standard, research and development of an embedded Java platform was performed. This required porting a Java virtual machine to run on an embedded microprocessor. An existing industrial network protocol DeviceNet was chosen for distributing the data between the network of control devices. To achieve this an upgrade was required to an existing DeviceNet communications engine to support distributed control. A third party IEC 61499 software application engine was ported to run on an embedded microprocessor. This option was chosen rather than completely developing a software engine as a commercial decision by the developer company. It also allowed support from other companies and researchers working with this standard. To test distributed control using this function block programming standard a test application consisting of a conveyor and three axis robot was developed. The test application demonstrated the feasibility of distributed control using IEC 61499 function blocks and some of the advantages of distributed control. Further outcomes of this research have highlighted some of the problems that require rectifying before this function block programming standard is feasible for commercial products.

# Acknowledgments

I wish to thank TCS (Tait Control Systems Ltd), the developer company for supplying the research project, and the Foundation for Research and Technology for funding. The following staff members of the developer company are acknowledged for help and assistance throughout the research project; Peter Tait, Nathan May, Bernard Wood, Aaron Wilson, and Alistair Edgar for assistance with hardware and software development. I thank Tony Dawson of TCS for handling paper work required for external funding proposals.

I wish to thank Dr Jim Christensen of Rockwell Automation for his past research and assistance with IEC 61499 technology.

I wish to thank Dr Frans Weehuizen my supervisor at Massey University and Nathan May my supervisor at TCS.

I wish to thank Annelese Blackbourn for assisting with proof reading.

I thank the following companies and tertiary institutions for supplying hardware and software for this research.

- Sun Microsystems for their Java virtual machine.
- Rockwell Automation, for supplying their IEC 61499 run-time environment.
- Waikato Institute of Technology, for supplying the conveyor and three axis robot machinery used for the test bed application.

The following trademarks used in this report are listed below with their owners:

- DeviceNet - ODVA (Open DeviceNet Vendors Association inc).
- EtherNet/IP - ODVA.
- RSNetWorx - Rockwell Software Inc.
- IEC 61499 - International Electrotechinical Commission.

(Adapted from Meek, 2003b, p. ii).

# Contents

# List of Figures

# List of Tables

# Chapter 1 Introduction

At present a lot of factory control systems use centralised architectures with a PLC and distributed I/O. Distributed I/O is where I/O modules are remote from a central controller and distributed on a network. With a centralised control architecture there are some inherent disadvantages such as: single point of failure possible, a large central processor, proprietary programming interface, and use of significant network bandwidth. There is the view that distributed control will overcome some of these problems. Previous implementations of distributed control have had disadvantages, such as different interface standards, each device requiring individual programming, and no programming environment suitable for the skill level of those who program the control systems. There is the hope that the developing IEC 61499 standard will overcome some of the problems with both past and present technology. This is a standard that is currently under development and its use has been limited to academic applications and specification development. The aim of this research is to: Investigate the IEC 61499 specification (IEC TC65/WG6(PT1CD)4. 2003, IEC TC65/WG6(PT4PAS)FD. 2002) and build a device that is commercially acceptable to industry. The final objective was not attained, however it has highlighted the areas where further research is required. These are as follows:

- Specification evaluation: This was finding out if the IEC 61499 specification is mature enough for commercialisation.
- Research and development of a suitable hardware platform.
- Development of peer-to-peer communication. This utilised existing peer-to-peer communications software with DeviceNet, an automation field bus network to communicate with the distributed devices.
- Development of a prototype product to demonstrate IEC 61499 capabilities for commercialisation.

## Investigation of the Status of the IEC 61499 Specification

The investigation of the this specification was to:

- Find out if the specification has advanced enough to develop a product using this technology.
- To gain familiarity with this standard and the programming methodology.
- Find out what other research with this technology has been performed.
- Develop a project plan showing how the research would be performed.

## Research and Development of a Suitable Target Platform

The aim was to investigate and develop a target platform using function block programming. This required research in the following areas:

- What platforms other researchers have used.
- The programming language other researchers have used to develop their IEC 61499 run-time environments.
- Develop the necessary tools and platform required to complete this research.

## Research and Development of an IEC 61499 Run-time Environment

An IEC 61499 run-time was researched and developed. As a result of this research, it was decided to purchase Rockwell Automation's experimental IEC 61499 run-time environment and modify it to suit the developer's target platform and applications.

## Modification of Existing DeviceNet Peer-to-Peer Communication Software

Research and development of the DeviceNet peer-to-peer communication protocol has the focus of a previous research project. This is an existing industrial network protocol used in industry. For further references refer to (Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1). This required modification of the developer's communication engine so it was compatible with an IEC 61499 run-time environment. The previous project added peer-to-peer communications to the developer's DeviceNet communications engine.

## Development of an Application

During the process of research and development, application software had to be developed for testing the function block run-time environment, function block types and communication interfaces. The final result of the project was a demonstration of proof of concept, and as discussed in later chapters the requirements for commercialisation of this technology.

This research is being performed for a product developer Tait Control Systems Limited (TCS). Their field of work is in design and manufacture of factory automation products. These include DeviceNet communication interfaces for centralised distributed I/O control systems. Their aim is to further develop these interfaces allowing them to be end-user programmable and have distributed control functionality. The scope of this research is to investigate the feasibility and development of a prototype IEC 61499 product. The developer demonstrated a working model using this technology at the EMEX trade show at Auckland, May 2004 which created industry interest.

# Chapter 2 Overview and Literature Review of Function Block Programming

## Background to Factory Automation and Control

In the early days of the industrial revolution, factory processes required large amounts of human labour and intervention. As technology has progressed, systems have been developed to automate and reduce the amount of human labour and intervention required. Throughout the development of factory automation and control, different types of control systems have been developed, some of which have now been superseded. Some existing and past systems are; mechanical, pneumatic, electro-mechanical and programmable logic controllers (PLC's). At present the developer company uses a centralised system consisting of a PLC. In industry there is now a trend towards the next generation of factory control and automation, termed 'Distributed Control.'

## The Definition of Distributed Control

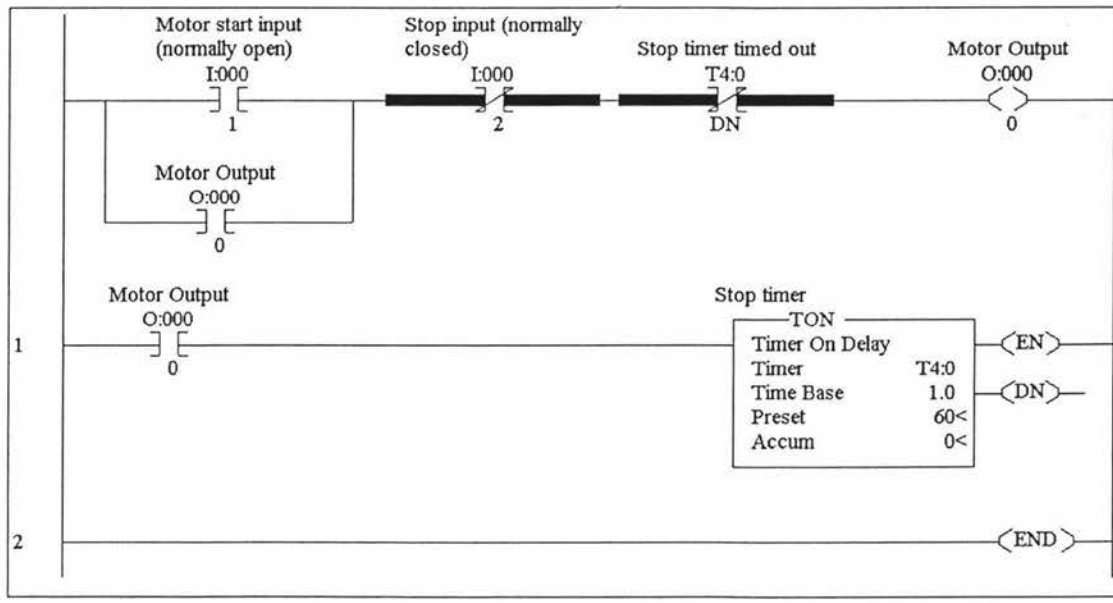
The following has been adapted from a previous research report written by the author. "The depth and extent of how the term distributed control is interpreted varies. A customer of the developer company has termed distributed control as a system used to distribute interface ports around their factory, thus reducing wiring looms. This type of control is distributed I/O. 'A distributed architecture based around DeviceNet became the ideal solution. With DeviceNet, Carter Holt Harvey was able to simplify the wiring scheme, saving time and money.' (Open DeviceNet Vendor Assocation, Release 2.0 Eratta 5, 2001, p. 1). While Shin et al. as quoted by Yook et al. suggest that: 'Although the term 'distributed control' has been used to refer to a centralised control strategy with a distributed computing (multiprocessor) implementation, we will use the term 'distributed control' to refer to control systems with physically distributed processing

power and network communication.' (Yook et al., 2001, p. 59). This is the scenario that will be used for this project. Tait gives two descriptions of distributed control: 'Distributed Control is such that the control of the process is distributed around the nodes of a network removing the need for a PLC.' (P.C. Tait, personal communication, 2002). This is essentially the same as Shin et al. as quoted by Yook et al. This is the definition that will be used throughout this report. The second definition refers to a semi distributed type system, which he defines as Embedded Control: 'Embedded Control is such that some of the tasks of the PLC are transferred from the PLC to the node on the network however the node can only communicate with the PLC.' (P. C. Tait, personal communication, 2002). TCS are implementing an Embedded Control approach with their current product range." (Adapted from Meek, 2003b, p. 9-10).

## Industrial Programming Languages

Most current centralised control systems use 'Ladder logic'. This is a software language used for programming PLC's. As discussed in more detail later, it was developed to replicate relay-wiring diagrams. An example is shown in the figure below.

**Figure 2.1 Example PLC program**



The ladder program shown controls a motor with two discrete inputs, an 'On' and 'Off' button. The vertical lines on each side of the diagram represent power supply rails. The first rung (horizontal line) is a circuit that enables a motor run output when the 'On' button is pressed. The motor output is also used as an internal input that shorts the 'On' input out when the motor is running. When the 'Off' button is pressed the circuit is broken and the motor stops. Rung number 1 controls a timer which after the motor has been turned on, will activate input T4:0 on rung 0 and stop the motor.

"Ladder logic was developed in 1970's when the PLC replaced the relay cam timer. The use of ladder logic allowed skilled electrical staff to understand the functionality of the control systems as it was constructed in a way that replicated relay-wiring diagrams. Ladder logic has remained the programming language of choice for New Zealand control systems, and is generally proprietary and can be used if licensed by the associated PLC vendor." (Meyer, 2001, p. 9). As stated in the quotes above PLC ladder logic is generally implemented with a proprietary language that each vendor provides the use with their PLC. The quote below defines some of the problems with these proprietary languages. "Control applications have been developed in BASIC, FORTH, C, Structured English, Instruction List and numerous other proprietary languages including various dialects of Ladder programming. Unfortunately, the only thing that can be said of all of these programming languages is that they are all different. There is clearly a waste of human resources involved in training staff in skills in so many different control languages." (Lewis, n.d., p. 1)

The article of Lewis's explained that for these reasons the IEC 61131 suite of languages was developed. However "IEC 61131-3 has focused on standardising PLC languages for single processors or small configurations with a few closely coupled multi-processors" (Lewis, 2001 as quoted in Meyer, 2001, p. 12). With the move to distributed control a non-proprietary programming language is required which has facilities for distributed control. From past research performed by the developer, the IEC 61499 standard has been chosen.

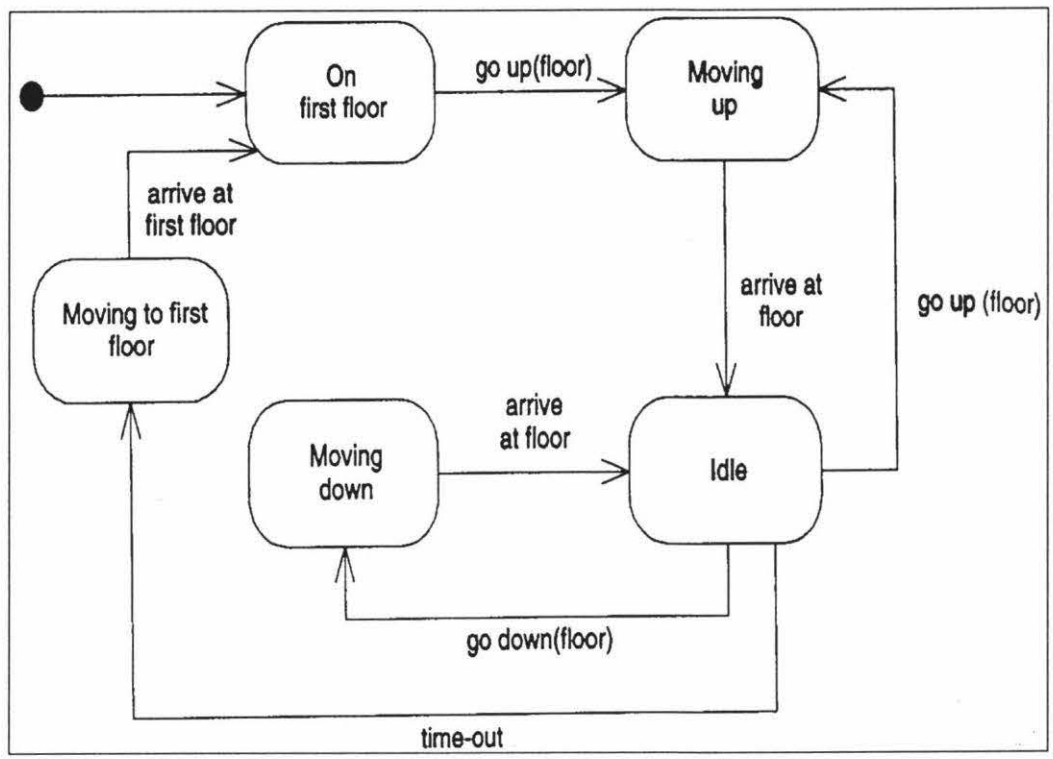
Searching for other distributed control systems, it was noticed that they existed, but there do not appear to be any standards other than the emerging IEC 61499 and IEC 61804 (IEC 61804 is discussed later on) for programming them. Other research on distributed control, shows that if IEC 61499 is not used then a proprietary system designed for the experiment is used. Examples of other systems are; "UDP/IP based distributed control for a hydraulic valve" (Heffernan & Murphy, 2003, p. 60). This is a system based on a network consisting of a PC and embedded microprocessors. C++ was the software language used. An example application using IEC 61499 function blocks is water level control as used by Wei (Wei, 2002, p. 36, 37).

The developer wants to avoid proprietary systems and systems that require a steep learning curve for the developer's potential customer base. When being briefed on this project Tait defined his reasons for using the IEC 61499 standard as:

- "IEC 61499 is a language which is designed to introduce object orientated principles to ladder logic based software programmers." (P. C. Tait, personal communication, 2003).
- "IEC 61499 will be a step towards educating existing ladder logic programmers to use an object orientated programming structure, such as UML. At present UML is considered too difficult for most existing ladder logic programmers." (P. C. Tait, personal communication, 2003).

When looking at Unified Modelling Language (UML) it is noted that it was a tool to assist with software design. UML is a modeling language that has been designed to assist software development. It has several levels of diagrams. The figure below is a state diagram. "Which is typically a complement to the description of a class. It shows all the possible states that objects of the class have, and which events cause the state to change." (Eriksson & Penker, 1998, p. 20)

**Figure 2.2 UML state diagram**




(Eriksson & Penker, 1998, p. 20)

The states are in the boxes with the rounded edges. The lines between the states represent the state transitions. The words associated with each transition are the event signals. Erikson and Penker state that: "The importance of models has been evident in all engineering disciplines for a long time. Whenever something is built, drawings are made that describe the look and behaviour of that "thing." The thing under development may be a house, a machine, or a new department within a company." (Eriksson & Penker, 1998, p. 1). They then proceed to explain that this situation applies to software development, and UML has been developed to standardise software modelling. "The goals of UML, as stated by the developers, are:

- To model systems (and not just software) using object-oriented concepts.
- To establish an explicit coupling to conceptual as well as executable artefacts.
- To address the issues of scale inherent in complex, mission-critical systems.
- To create a modelling language useable by both humans and machines."

(Eriksson & Penker, 1998, p. 5)

With UML being a modelling language it is not suitable for implementing distributed control systems. The modelling language only describes the application, but does not generate any code to run the application.

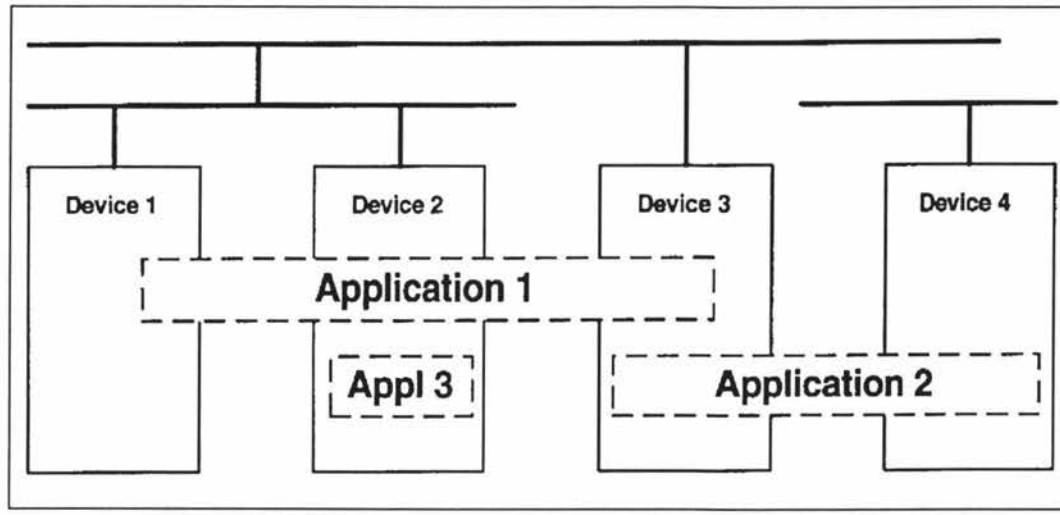
An alternative function block standard emerging is IEC 61804. The standard appears still to be in the development and conceptual phase, so the information available is not clearly defined. From literature the description of IEC 61804 is: "The main purpose of this part 1 of draft IEC 61804 is the harmonisation of different views, models and starting points of end users, system providers and device manufacturers." (Diedrich et al., n.d., p. 3). This implies that this standard is aimed at standardising interfaces between devices, field buses and configuration tools. Further literature defines IEC 61499 as a prerequisite. "A prerequisite to design, implement and operate a FB based process control system is that the tools and the devices follow the same architecture based on a common specification... The PAS IEC 61499 Function block model on which this requirement specification is related is able to provide these basic components for Function Blocks for process control." (Diedrich et al., n.d., p. 3). From the above, the IEC 61804 standard appears to be currently under development and is built around IEC 61499. Presently, literature does not show any use of the IEC 61804 standard.

## Introduction to IEC 61499 Function Block Programming Language

The IEC 61499 standard defines a function block programming language that is currently under development. "The IEC 61499 standard, which builds on function block concepts defined in the PLC language standard IEC 61131-3, is being developed in liaison with Fieldbus standardisation work" (Lewis, 2001, p. 5).

The top end model of an IEC 61499 program is known as the system model. This is shown in the figure below that illustrates the nature of resource sharing with distributed control.

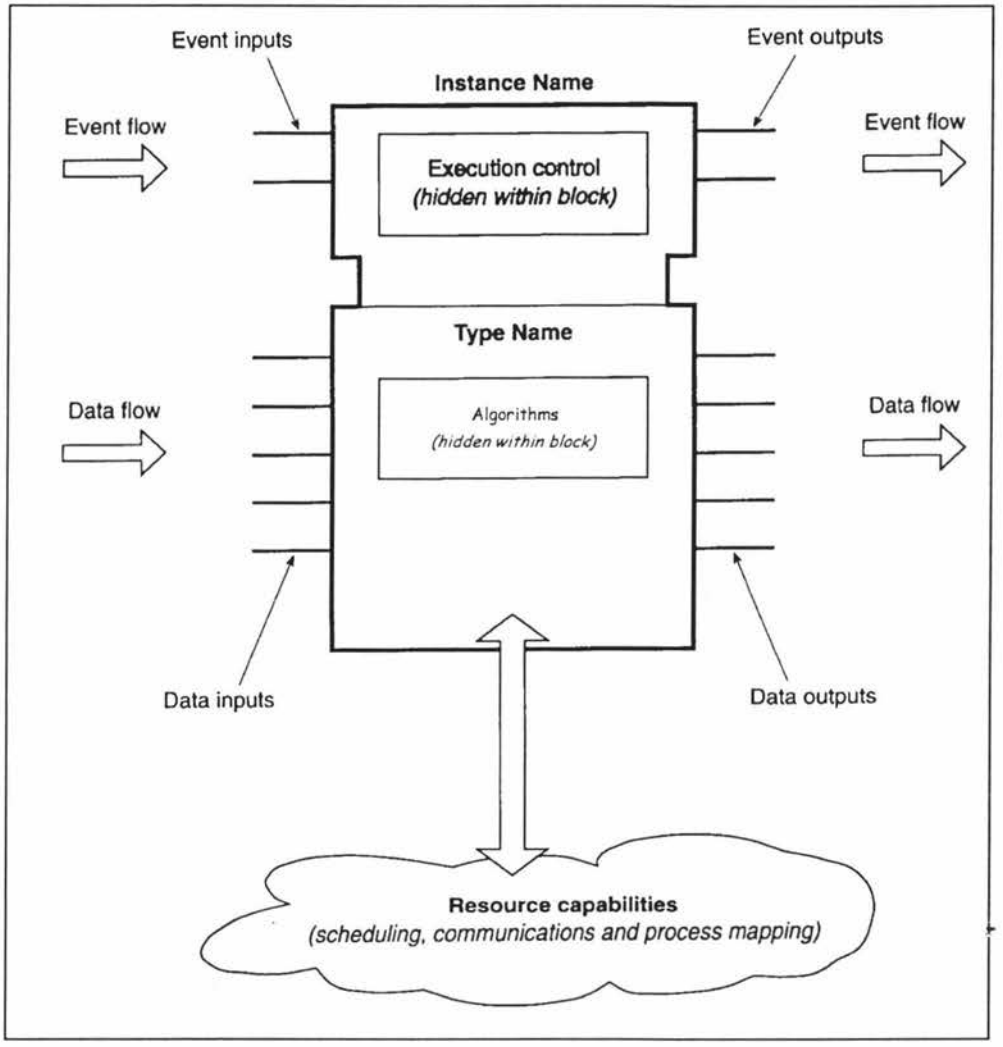
**Figure 2.3 Resource sharing over a network**




(Lewis, 2001, p. 28)

This shows distributed control applications which have their resources shared over multiple nodes on a network. The applications themselves are made up of function blocks where Lewis states: "A function block is described as a 'functional unit of software' that has its own data structure that can be manipulated by one or more algorithms." (Lewis, 2001, p. 26). The function blocks themselves are represented as shown in the figure below.

**Figure 2.4 Declaration of a function block**

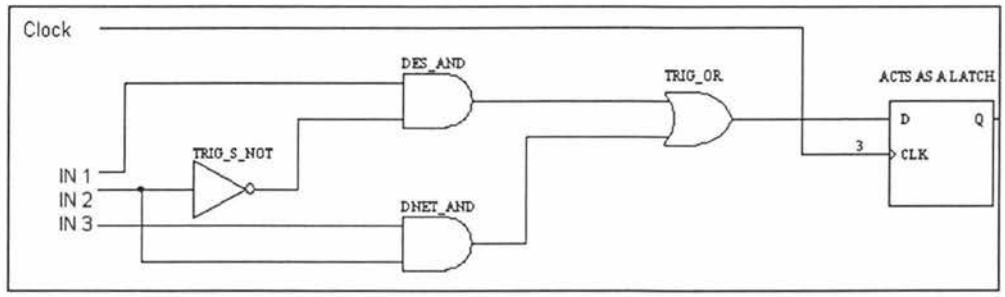



(Lewis, 2001, p. 22)

The internal components of a function block are algorithms. These will be described later in this chapter. The function blocks are connected together as shown in figure 2.6 by event and data links. These are shown as the event inputs and outputs, and the data inputs and outputs. The data inputs and outputs are the links through which data is transferred. The event inputs transfer the event data, which acts as the strobe pulse for the data transfer links. Different data links can have different event links for transfer of data. An event link does not necessarily have to have a data link associated with it. As an example, an initialisation event may only initiate the internal algorithms inside the device. Execution control and the algorithms themselves are explained further down.

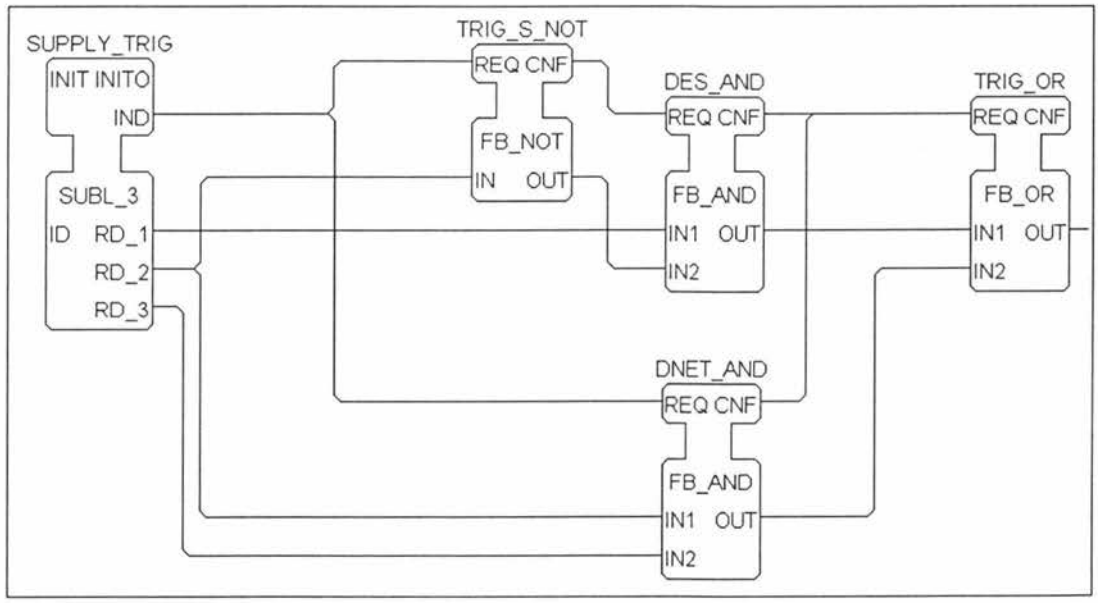
The figure below represents an example application using logic gates.

**Figure 2.5 Representation using logic gates**




The figure below is a function block representation of the above application.

**Figure 2.6 Example function block application**




The function block program is a group of logic functions.

TRIG_S_NOT is a function block that is the equivalent of a NOT inverting logic gate.

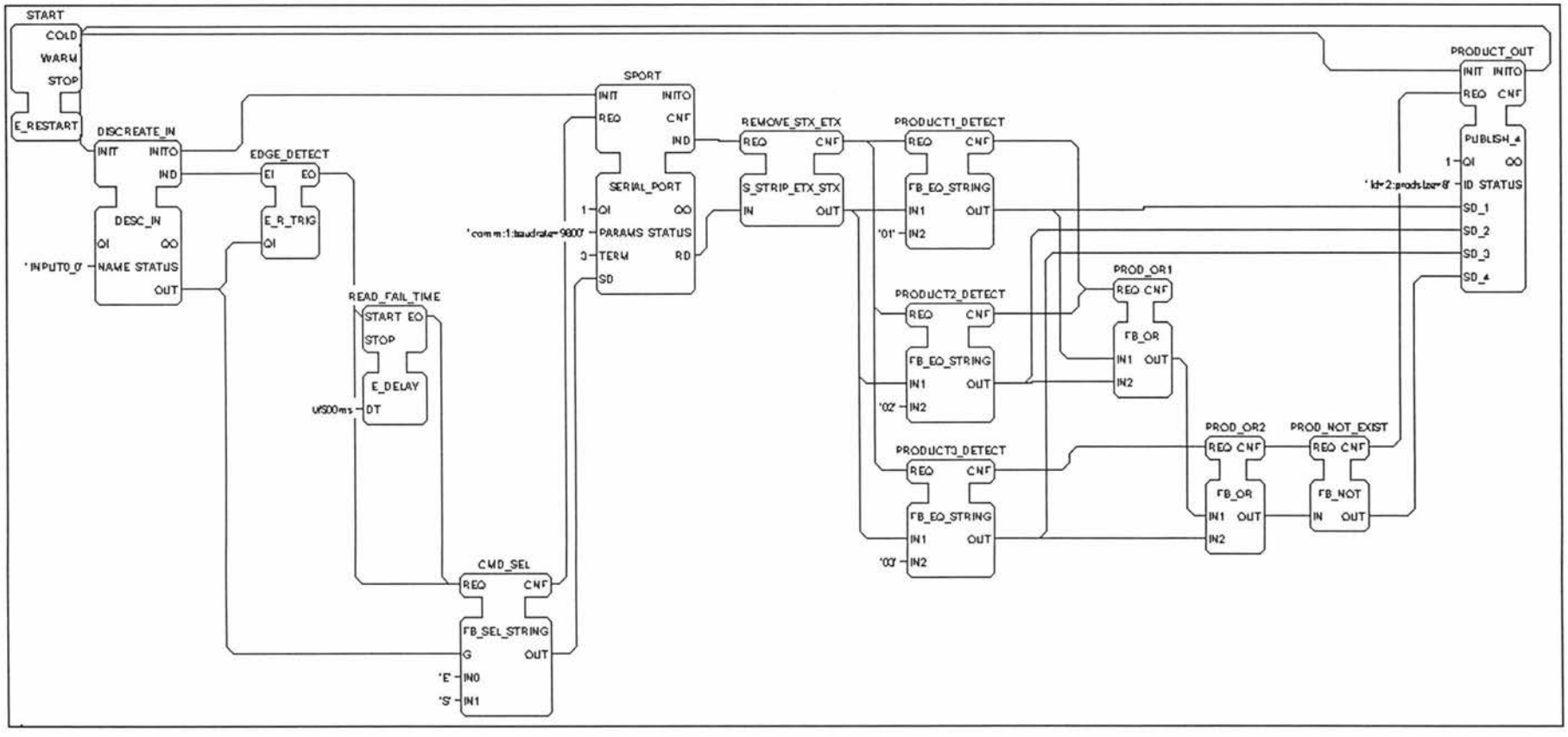
DES_AND and DNET_AND are the equivalent of AND logic gates.

TRIG_OR is the equivalent of an OR logic gate.

All function blocks have an event input that acts as a clock strobe.

A more complex application demonstrates the difference between a ladder logic program and the equivalent function block program. The function block program is discussed in chapter 6 and is shown in figure 6.19. The function block program interfaces a bar code scanner, checks for three bar codes and activates an output via a network interface for each.

**Figure 2.7 Bar code scanner function block application**



**DESCRETE_IN** – reads in the discrete input to trigger the bar code scanner read. This is the input from an external digital input. It has three output connections, one is for indicating that initialisation is complete, one an indication of input change of state and the other the value of the input.

**EDGE_DETECT** – detects a rising edge from the discrete input. This detects if the change of input state is a rising edge.

**READ_FAIL_TIME** – is a timer that cancels the bar code scanner read if the bar code scanner has not responded to the 500ms. This timer times the bar code scanner out.

**CMD_SEL** – selects the command that is sent to the bar code scanner. 'E' cancels the read, 'S' initiates a read. The 'S' and 'E' are ASCII characters that are sent out of the serial port.

**SPORT** – the serial port interface to the bar code scanner. It has a data and event input to transmit serial data and a data and event output for indication of received data.

**REMOVE_STX_ETX** – removes the start and end characters from the serial communications string received from the bar code scanner.

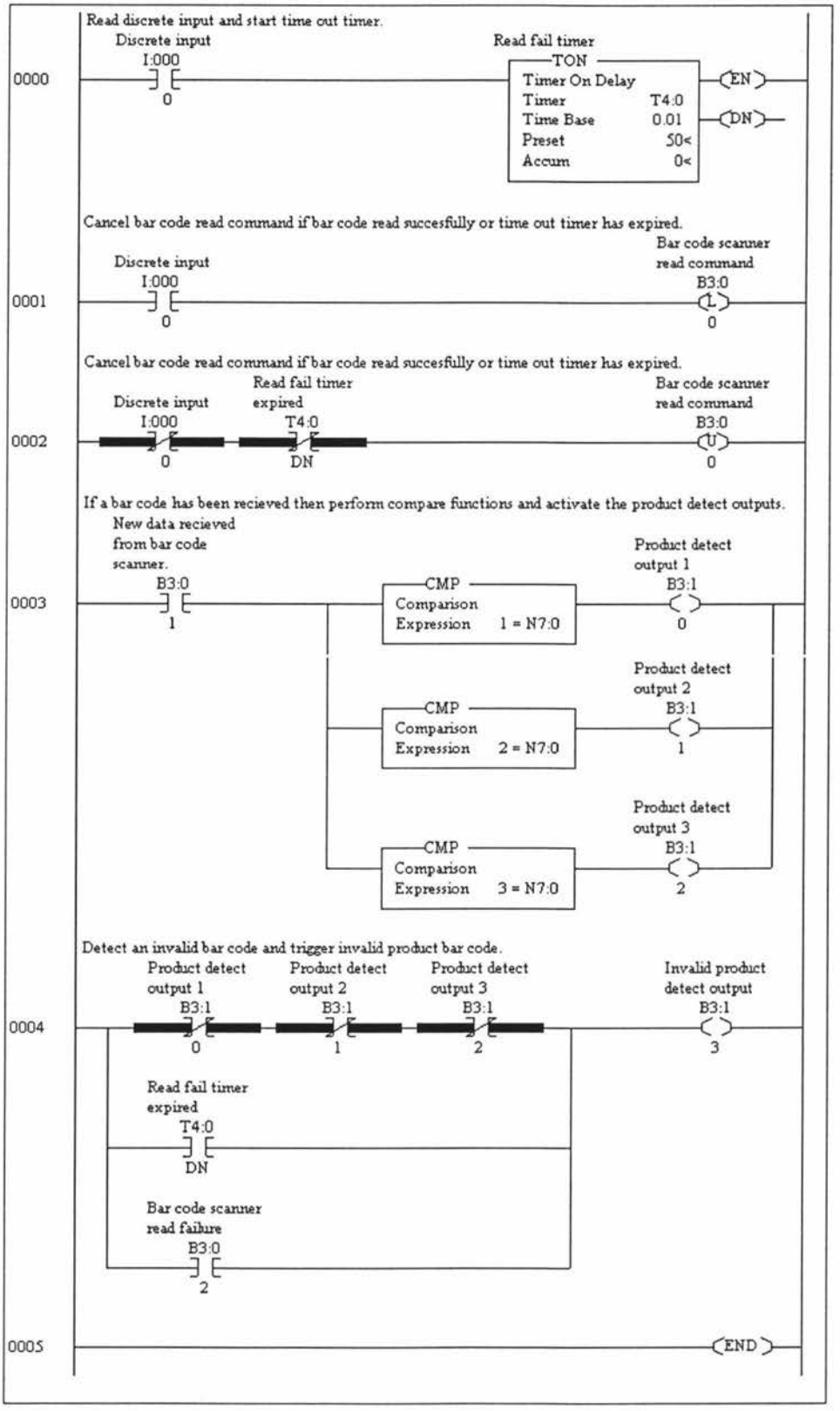
**PRODUCT1_DETECT, PRODUCT2_DETECT and PRODUCT3_DETECT** – compare the received bar code string and perform the sort operation.

**PROD_OR1, PROD_OR2 and PROD_NOT_DECT** – are for detecting an invalid bar code read. This is the instance where the 'PRODUCT_DETECT' function blocks do not detect a valid bar code.

**PRODUCT_OUT** – is the network interface that the bar code result is sent to.

To achieve the same functionality with ladder logic a similar ladder diagram would be required.

## Figure 2.8 Equivalent ladder logic program



For this application the equivalent ladder logic program may not look complex, however the function block diagram shows a graphical layout illustrating how the program operates.

## Internal Operation of Function Blocks

There are the following types of function blocks:

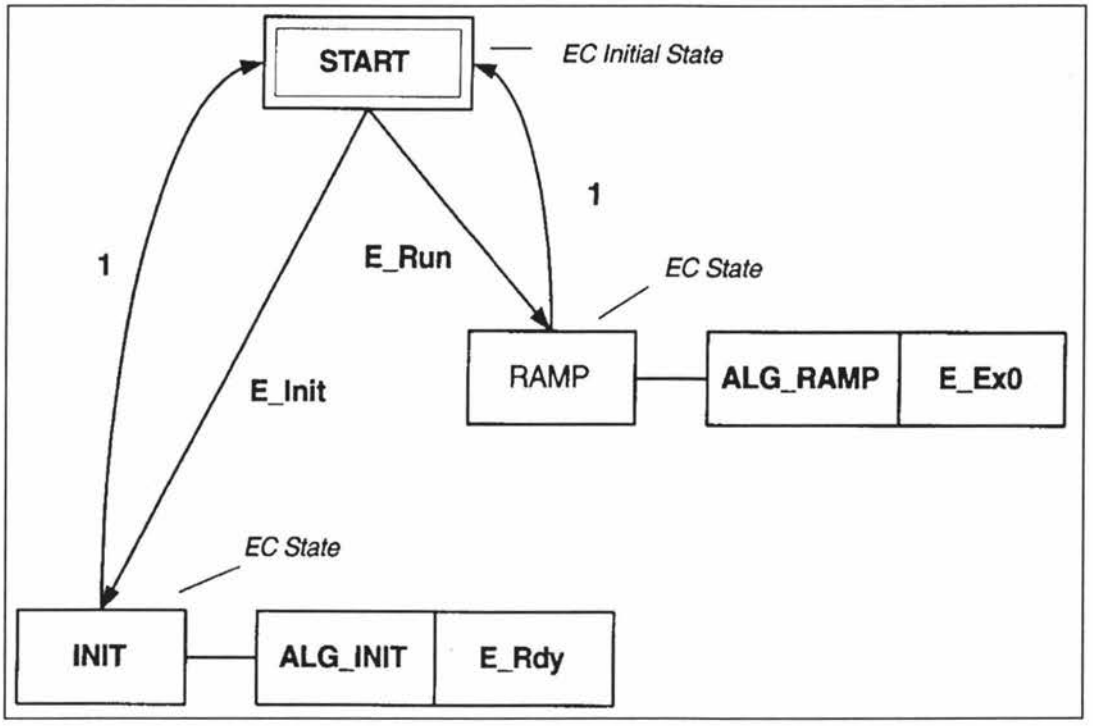
- Basic function blocks.
- Composite function blocks.
- Sub-application function blocks.
- Adaptor interfaces.
- Service interface function blocks.

### Basic Function Blocks

The basic function block is defined in figure 2.4. There are two main components inside the function block: the execution control chart and the algorithms. The execution control is handled by a state machine termed an 'Execution Control Chart' (ECC). Lewis states: "The ECC is primarily intended to represent the relationships between input events, algorithm execution and the firing of output events." (Lewis, 2001, p. 49)

A sample execution control chart for the ramp controller function block is shown below.

**Figure 2.9 Example execution control chart**




(Lewis, 2001, p. 48)

The execution control chart has an initial state of 'START.' When the function block receives an event input, it will change state. The event inputs that can cause a change of state are 'E_Init', 'E_Run' and '1'. If the 'E_Init' event occurs then the ECC will change to the 'INIT' state and execute the algorithm 'ALG_INIT.' Upon successful execution of the algorithm the 'E_Rdy' event output will be set. The event input of '1' means that once the algorithm is executed return, to the 'START' state. The same procedure happens for an 'E_Run' event and the 'RAMP' state.

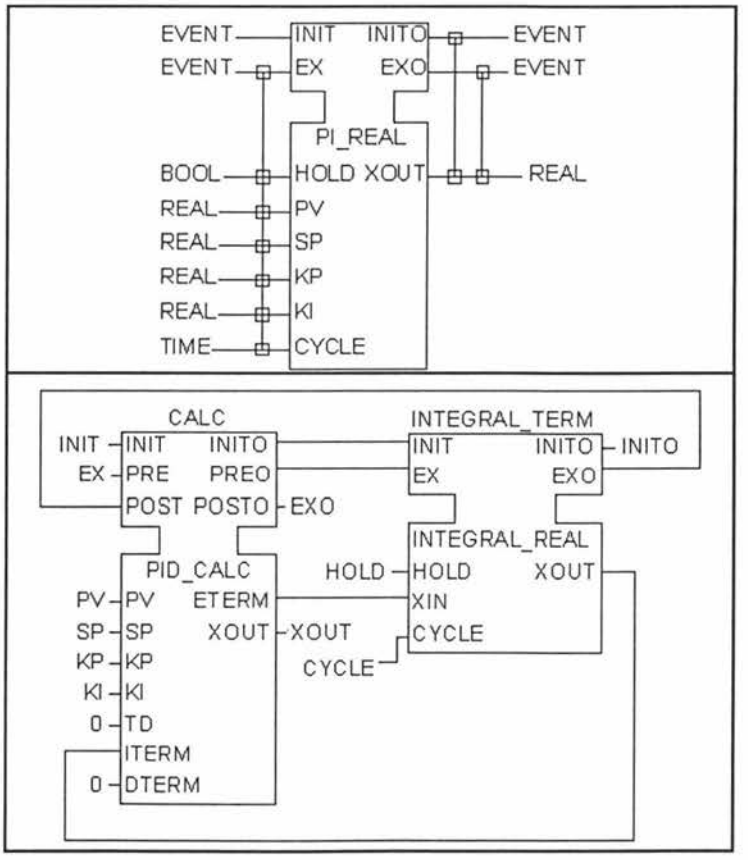
When the execution control chart transitions to another event an algorithm may be executed. The algorithms perform the tasks of the function block. The IEC 61499 standard proposal does not define any language for the writing of algorithms. Lewis indicates that "IEC 61499 does not preclude the use of other languages such as Java or C to define the algorithm contents." (Lewis, 2001, p. 41). Evaluating the IEC 61499 editor software which is an experimental prototype function block programming

software package, it supports the following programming languages: ST (Structure Text), LD (Ladder Diagram), FBD (ST, LD and FBD are all defined by IEC 61131) and Java. Correspondence with Dr Jim Christensen has indicated that C++ has also been used (J. H. Christensen, personal communication, March 5, 2003). Dr Jim Christensen is the researcher of the IEC 61499 standard at Rockwell Automation.

## Composite Function Blocks

Lewis gives the following description: "Composite function blocks provide a means for building up more complex blocks from basic and other smaller composite blocks in a hierarchical fashion." (Lewis, 2001, p. 55). Composite function blocks do not have any execution control charts or algorithms itself, as it is constructed from basic function blocks.

**Figure 2.10 Example of a composite function block**



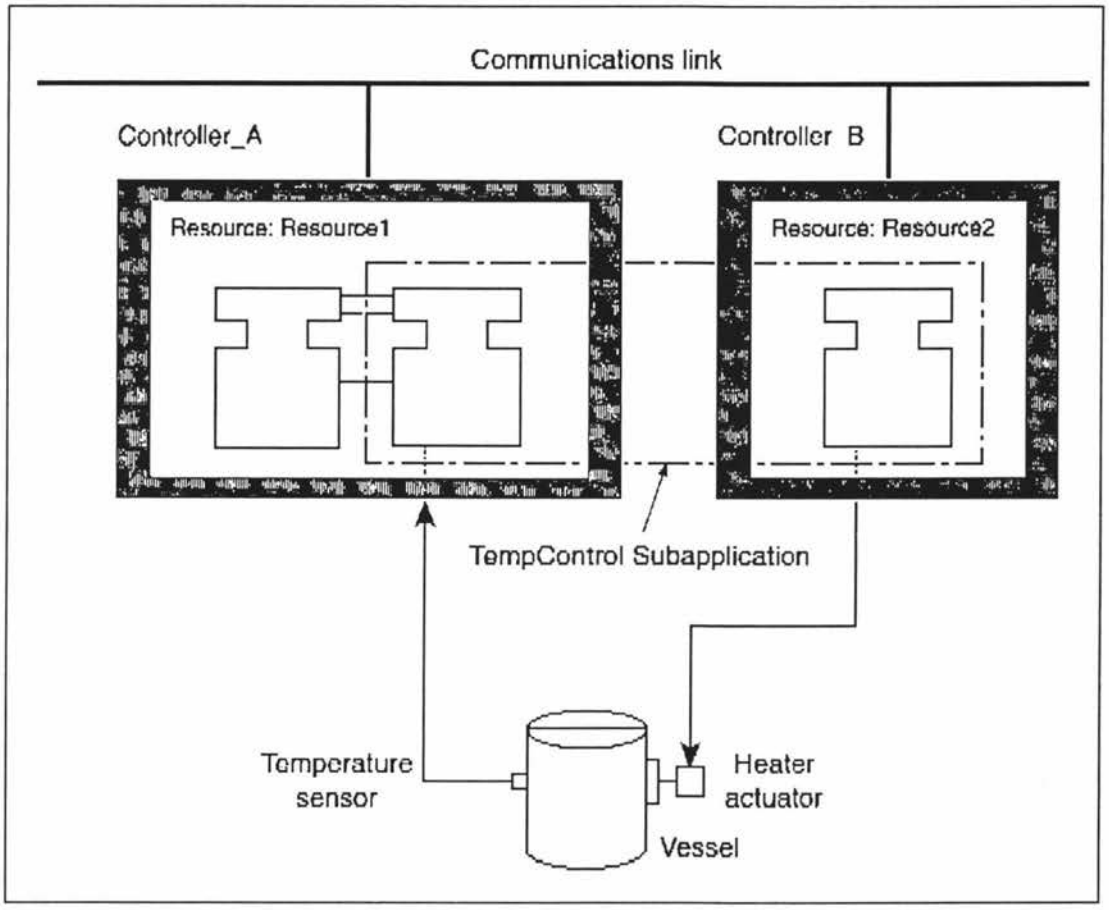

(IEC TC65/WG6(PT1CD)4, 2003, p. 35)

The figure above shows an example of a composite function block. The top half of the figure is the composite function block whereas the bottom half shows the individual function blocks that build up the composite function block. The diagram shows the event and data links between the two function blocks. The labels outside the function block correspond to the external connections of the combined function block.

## Sub-application Function Blocks

A sub-application function block is similar to a composite function block. Lewis explains: "The main contrasting feature of a sub-application when compared with a composite function block is that it can optionally be run on multiple resources. Remember that basic and composite blocks can only run on the same resource, ie it is not possible to break them down into parts that can run on different resources." (Lewis, 2001, p. 61). Examples of IEC 61499 programs show that a resource is an interface to some type of communication system. The figure below shows an example of an application that has two controllers, which has a sub-application distributed between them.

**Figure 2.11 Example of distribution with sub-applications**



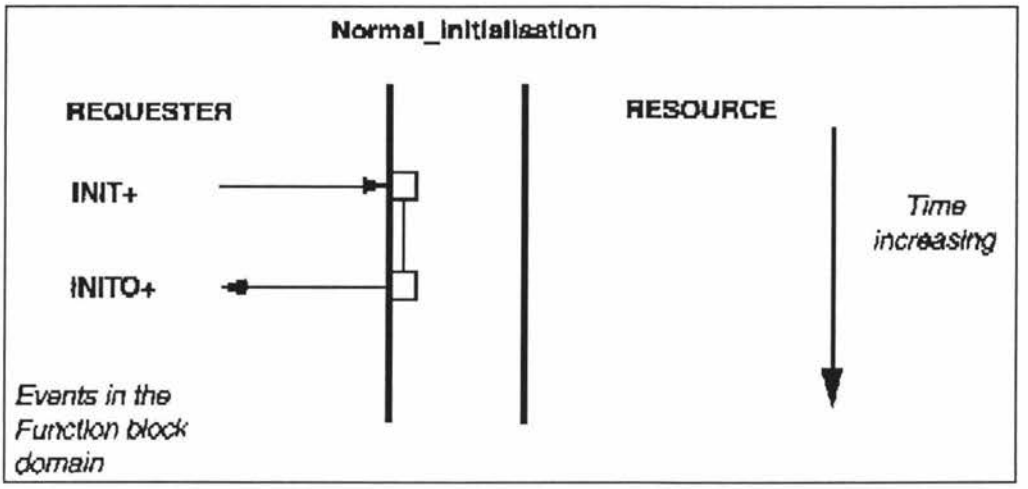

(Lewis, 2001, p. 66)

## Service Interface (SI) Function Blocks

The previous function blocks except sub-applications that have been discussed are used for performing internal functions where no outside communications are required. Service interface function blocks are used: "Wherever any form of interaction is required between function blocks within the resource and the external world, there is a requirement for an SI function block." (Lewis, 2001, p. 69). In appearance and operation a SI function block is similar to the basic function block. The event and data inputs/outputs have defined data types to build a standard. The internal algorithms communicate with outside input and output devices. This uses a time sequencing

diagram in place of the execution control chart. An example of a time sequence diagram is shown below:

**Figure 2.12 Example time-sequence diagram**




(Lewis, 2001, p. 78)

The requestor side shows the event inputs. An INIT+ event is the initialisation event. When a request for initialisation is made, the resource is initialised. When initialisation of the resource is complete an INITO+ event is generated.

## Construction of a Function Block System

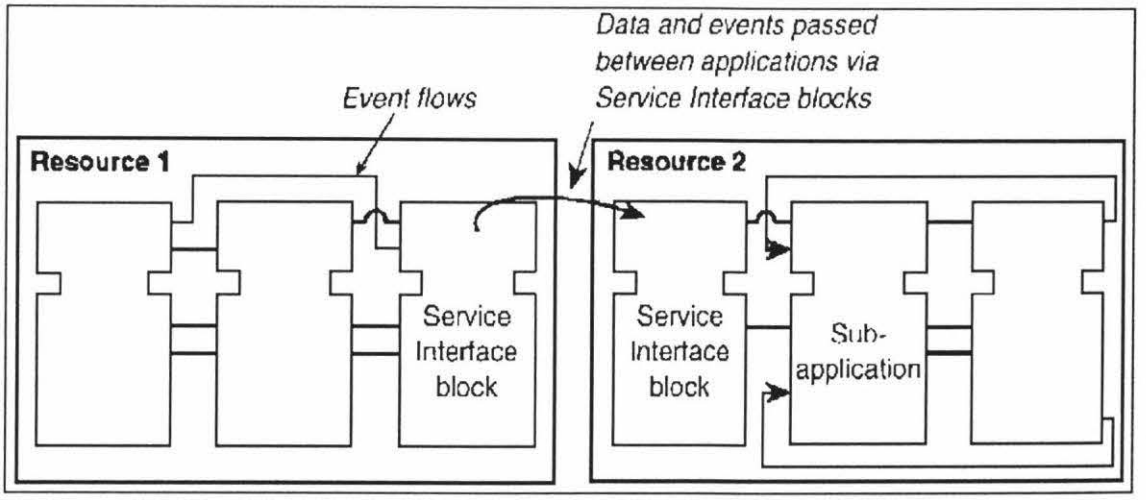
A whole function block distributed control application, is termed a 'System.' Lewis describes this: "At the physical level, a distributed system consists of a set of devices interconnected by various networks to form a set of co-operating applications. An application such as the control of a production line, process vessel, and conveyor will typically require the interoperation of software running in a number of devices." (Lewis, 2001, p. 22). Figure 2.3 is an example of a system consisting of multiple applications shared amongst multiple devices.

As shown in figure 2.3, a device is the individual node on a distributed control network across which the applications are distributed. The IEC 61499 specifications are unclear as to what the different types of devices are, so the list below is only illustrative:

- I/O modules.
- Serial interfaces.
- HMI (Human Machine Interface) interfaces.

The specification states that: "A device shall contain at least one interface, that is, process interface or communication interface." (IEC TC65/WG6(PT1CD)4, 2003, p. 17). This implies that a device has either interfaces to I/O functions and/or to a communications network. The devices themselves are constructed resources. Lewis states that: "The resource provides facilities and services to support the execution of one or more function block application fragments." (Lewis, 2001, p. 24). The specification states: "A resource is considered to be a functional unit, contained in a device which has independent control of its operation. It may be created, configured, parameterised, started up, deleted, etc., without affecting other resources within a device." (IEC TC65/WG6(PT1CD)4, 2003, p. 17)

**Figure 2.13 Distribution over resources**




(Lewis, 2001, p. 26)

The above figure shows an application that is spread over two resources. Each resource is part of the program like a subroutine, but uses a service interface function block to transfer data between the two resources. "Fragments of function block networks are distributed between resources that exist either on the local device or in resources on remote devices." (Lewis, 2001, p. 24). This implies that resource sharing within an

application is from a combination of resources within a device or amongst multiple devices.

## Software Implementation of the IEC 61499 Standard

This section describes the software internals of function blocks. The specification defines three classes of devices:

- "Class 0: Simple Devices
- Class 1: Simple Programmable Device
- Class 2: User-reprogrammable devices" (IEC TC65/WG6(PT4PAS)FD, 2002, p. 8)

Interpreting table A.1, Class 0 devices have function block instances pre-loaded into them. The only programming capabilities are the creation of connections between function blocks. Class 1 has all the capabilities of class 0 with the capabilities to create function block instances. "Devices in this class will typically have a predefined set of function blocks, for example, held in firmware." (Lewis, 2001, p. 134). Class 2 has all the capabilities of class 0 and 1. In addition, these devices support the ability of having additional function block definitions loaded into them. "Such devices must allow new function block type definitions to be downloaded across a network from a configuration tool." (Lewis, 2001, p. 134).

With the description of the device types available, for a class 0 and 1 device, the manufacturer predefines the function block definitions. This allows the manufacturer to predefine the functional operation of the function block. As stated earlier, the specification does not define any language for writing algorithms. However it was observed using the Function Block Development Kit (FBDK the IEC 61499 function block development software available form www.holobloc.com) and discussed in "Modeling control systems using IEC 61499 : Applying function blocks to distributed systems" (Lewis, 2001) that there was a strong bias to the following languages; Structured text (ST), Function block diagram (FBD), Ladder diagram, (LD) and Java.

Even although the specification is language independent an investigation of their characteristics was considered important. ST, FBD and LD are all defined in IEC 61131 as follows:

ST (Structured Text), is a high level language. John and Tiegelkamp state that "PASCAL and C are comparable high-level programming languages in the PC world." (John & Tiegelkamp, 1995, p. 111). To use ST as a programming language for function blocks, a compiler would be required for the microprocessor used in the device. This may require developing a specialised compiler.

Lewis defines FBD as: "A graphical language for depicting signal and data flows through function blocks re-useable software elements. FBD is very useful for expressing the interconnection of control system algorithms and logic." (Lewis, n.d., p. 4).

Lewis defines LD as: "A graphical language that is based on the relay ladder logic, a technique commonly used to program the current generation PLCs." (Lewis, n.d., p. 5).

Java is the language that the Function Block Development kit uses for definition of algorithms. In email correspondence with Dr Jim Christensen he said that "To support any dynamic management features of IEC 61499 Class 1 and Class 2 devices, e.g., FB instance creation and deletion, is natural with Java." (J. H. Christensen, personal communication, March 5, 2003). Christensen is one of the main developers of the IEC 61499 standard. Java is run via a Java virtual machine and supports full object orientation. Research performed by Edgar has indicated that there is a Java virtual machine that has been developed for embedded systems. However as stated by Edgar "The interpretation at run-time makes Java generally run slower, and consume more memory than a compiled language like C." (Edgar, 2003, p. 2).

C and C++ are languages that could be used for function block definitions. The use of C, C++ or any language is not defined in the draft specifications, however Lewis suggests its use. "IEC 61499 does not preclude the use of languages such as Java or C to define the algorithm contents." (Lewis, 2001, p. 41).

## Summary of the Programming Environments

The PLC was developed to replace the mechanical and electromechanical control systems. Ladder logic was developed as a simple language to allow those who were less skilled with electronics and software programming to write software for a PLC. The IEC 61131 standard was developed to standardise the different programming environments that have been used with factory control using a function block approach. The IEC 61131 standard was not designed to facilitate the development of distributed control architectures, where the PLC is removed from the control network and the control is distributed amongst the field devices. The IEC 61499 standard is being developed to overcome this problem and should introduce object orientation to those who are not familiar with it.

## Status of the IEC 61499 Standard

The IEC 61499 standard is still in the draft phase and at commencement of this research the developer company was the first to try and use it for a commercial approach. At present Rockwell Automation and others have been developing the standard and performing evaluation tests to test its feasibility. The software tools that are available are as follows:

- FBDK, "Function block development kit" from Rockwell Automation, this software is the IEC 61499 function block programming software. This is commercially available.
- FBRT, "Function block run-time environment", software that runs IEC 61499 programs on the PC. This is commercially available.

- CORFU FBDK, this software is from the University Patras in Greece and is a function block development tool similar to Rockwell Automation's version. This software is available to the public upon request.
- FBRT for embedded platforms. This is the same as the FBRT for the PC, but has been modified so it will run on an embedded platform, such as the SNAP. The SNAP is an embedded microcomputer made by Imsys. This software was made available for this research project through a licence arrangement with the sourcing company.

With the software tools available and the stage that the standard has been developed to, it is mature enough to start developing an entry level IEC 61499 based product.

# Chapter 3 Selection and Evaluation of Target Software Environment

## Status of IEC 61499 Development at Commencement of the Project

The IEC 61499 standard is a function block programming language for factory distributed control. At commencement of this project the standard was in the final conceptual stages of development. The tasks required to set up a development platform for the project are:

- Investigation of a hardware and software platform suitable for function block development. This includes the choice of programming language for use with software development.
- Configure the chosen platform for embedded function block development.

## Choosing a Suitable Programming Language

This research was performed for a product developer that has an ongoing research programme for developing distributed control products. With the developer's ongoing research and development programme, consideration is required for future development beyond the scope of this research project. When this project was initially proposed C was the language suggested for the run-time environment development. The initial stages of research were to investigate and become familiar with the function block architecture. This consisted of reading the specifications, becoming familiar with the 'Function Block Development Kit' (FBDK). This is a programming environment for programming and investigating function block applications. Discussions and email correspondence with Jim Christensen have shown that he and Rockwell Automation have used Java for the run time development. The Rockwell Automation run time environment executable is publicly available and can be expanded by writing new

function blocks with Java. The Technical University of Vienna has used C++ for development. (J. H. Christensen, personal communication, March 05, 2003). The University of Batras of Greece has developed their function block development kit with the intention of using a run-time environment on the PC. The language they are using is not stated. An investigation was performed with the following languages to see which would be most suitable.

The following languages were considered as to their suitability for use in this development project:

- C
- C++
- Java

C is a procedure-based language and its use is wide spread amongst small microprocessor applications. C++ is an extension of C but includes a superset of commands that make it object orientated. It is not as wide spread as C in small micro controller based applications as it uses larger amounts of memory and requires greater processor speed. Java is an object-orientated language that is compiled to a machine code known as a byte code. The byte code is executed by a virtual machine that acts as a cross platform emulator. As Java has a standard byte code it is portable between different microprocessor platforms. The language has been designed so the virtual machine looks after memory allocation and multi-threading. However this structure places large overheads on the microprocessor in terms of execution cycles and memory usage, therefore it is not so widely used for embedded applications.

Initially C was preferred because the existing products in the developer's product range use C. Java was considered because the developer wants to produce products that are interoperable with third party equipment vendors who are collectively developing the technology. This applies to future revisions of the function block run-time environment

that may include features such as downloading further function block definitions into firmware. The IEC 61499 specifications defines three types of devices:

- "Class 0 - Simple devices. For a device to be compliant with class 0 it should provide basic functionality to support; the creation of connections between function blocks, applications, and an external query to provide definitions of function block external interfaces.
- Class 1 - Simple programmable devices. This class of device has additional functionality beyond class 0 that includes the ability to create new function block instances as well as connections. Devices in this class will typically have a predefined set of function blocks, held in firmware; i.e. there is no support for changing on-board function block type definitions.
- Class 2 - User-programmable devices. In addition to the functionality of class 1, these devices have the ability to create new data types and new function block types." (Lewis, 2001, p. 133-134).

For this project the scope has been limited to class 1 devices. The reason for this is, the developer requirements do not extend to class 2 devices at this point in time. However, in future the developer may develop these devices. For this class device interoperability is an issue.

Looking at other research projects and the IEC 61499 specification, there was no indication that Java was a requirement for IEC 61499 run-time development, however in most cases there was an indication that they have used Java. As part of developing an understanding for function block concepts, a Function Block Development Kit (FBDK) software package developed by Rockwell Automation was evaluated. The FBDK is a graphical software programming tool that is used for writing function block programs. This software was written by Dr Jim Christensen of Rockwell Automation for IEC 61499 researchers to evaluate the standard. During the research phase of this project the FBDK was the only software tool available to write IEC 61499 compatible programs. www.holobloc.com is the web site were the FBDK can be downloaded. This software

has been written in Java, and all its facilities for adding further function block definitions are all Java based. After a further investigation for evidence to confirm whether Java is a requirement for the developer to have interoperability, the following information was found: "IEC 61499 does not preclude the use of languages such as Java or C to define the algorithm contents." (Lewis, 2001, p. 41). From correspondence with Christensen it was found that there was, "no real requirement for Java, although dynamic creation and deleting of FB instances and connections may be a challenge in C." (J. H. Christensen, personal communication, April 29, 2003). This confirms that Java is not a requirement for interoperabilty

The following also needed consideration:

- Whether the chosen language will limit the path of the developer company's future development.
- Whether the chosen language will be suitable for programming this application.
- Whether the cost of processor resources will outweigh the cost of what the product can be sold for in the long term.
- The cost of equipment required for development.

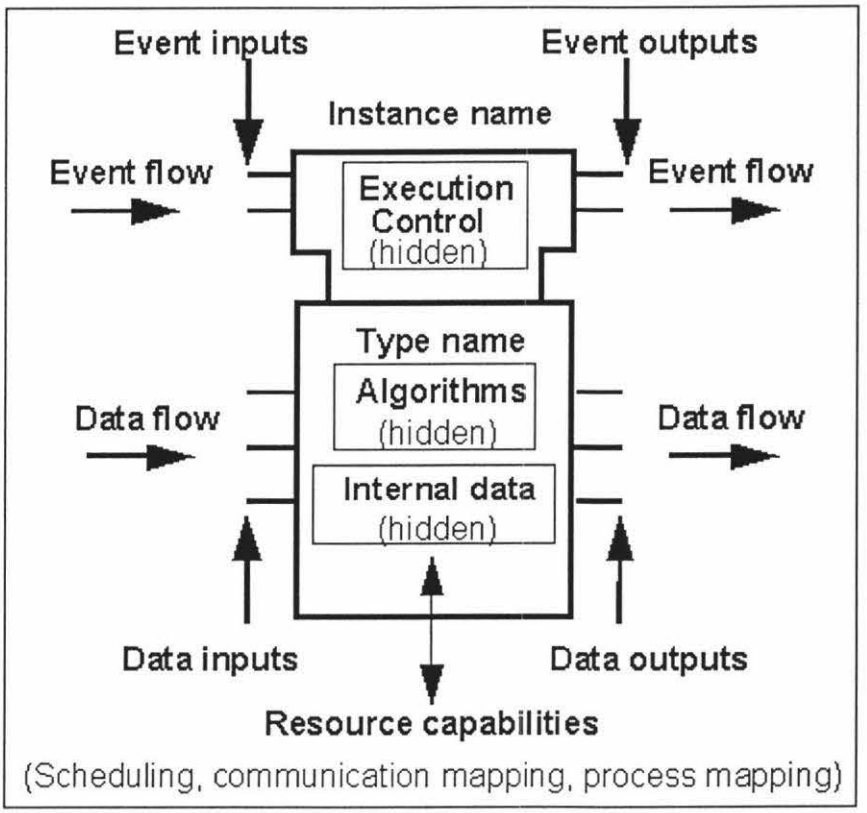
At commencement of this research project the developer was of the opinion that C should be used. The reasons for this decision were:

- In the past all their embedded equipment development has used C.
- Existing communication software engines have been developed in C and these are to be interfaced to the IEC 61499 run-time environment.
- Their hardware platform appears to have sufficient resources, (speed, RAM and ROM) to run C applications. However it looks unlikely that it would run a language such as Java.
- The developer already has the necessary development equipment, and expertise for developing in C.

- The project seemed achievable using this language.

To explain the areas where investigation is required, some background information about the IEC 61499 standard is given. From a personal discussion with the developer: "IEC 61499 is a language which is designed to introduce object orientated principles to ladder logic based software programmers." (Peter Tait, personal communication, 2003). The figure below is used to explain the object orientation of the standard.

**Figure 3.1 Basic Function Block**




(IEC TC65/WG6(PT1CD)4, 2003, p. 20)

This illustrates a function block, which is effectively a class. A function block is a unit of software that looks after a certain task. The class contains an Execution Control Chart (ECC), which is a state machine that handles the execution of algorithms. The algorithms are implemented in software creating the tasks of the function block. The data are the internal variables that are used by the algorithms. Each time this function block (class) is used, an object of that function block (class) is created. It is possible,

and highly likely, that the internal variables for each object are only specific to that object and will not be shared with other objects. With this structure, use of an object orientated language would make it simpler to develop. As discussed in previous sections, research by others has shown that they have chosen to use an object orientated language, such as C++ and Java. However other researchers have used a PC based platform which contains a large amount of processor resources, thus making it easier for them to use a language like Java. The processor platform the developer wants to use is an embedded one with fewer resources available.

As explained above, the IEC 61499 standard is object orientated, this requiring dynamic creation of objects, while C is a procedural language. While object orientation is not impossible in C, development can be difficult as dynamic memory allocation and pointers would be required. The latter can create potential memory leak problems, which can be hard to debug. Another reason for thinking C is not suitable is, the developer requires all code to be written in a modular structure making it simple for future additions. With careful programming this is possible with C, however object orientated programs lend themselves to this programming structure. When considering C, one of the concerns was that if the run-time was developed in C, the program might be pushed to its upper limits when adding future enhancements. The biggest issue for the developer was that the wrong choice of software language would be costly and time consuming if discovered at a point further on in research and development.

It was decided that an object orientated language should be used for the following reasons:

- The final outcome for research is to develop an object oriented run-time environment for the IEC 61499 standard.
- Properly designed object oriented code lends itself to modular software development.

The two object oriented programming languages considered were C++ and Java. C++ has a benefit over Java as it is compiled code where, as shown in figure 3.2 Java requires a virtual machine for interpretation. Unfortunately the requirement of the virtual machine can place a large burden on the microprocessor resources, because it is more an interpreted language. "It's been reported that Java is 10 to 20 times slower than C, on average." (Wickham, n.d., p. 1). However C++ like C is a compiled language. Upon considering compatibility of compilers and hardware platforms, it was found that at present most microprocessors that have the resources required for the developers current product range have a C compiler available. At commencement of the project there was doubt as whether the microprocessor used by the client has the capability to run the IEC 61499 run-time software. This may result in the developer upgrading to a more powerful microprocessor. For this reason the language that is chosen needs to be transportable between different microprocessor platforms.

For embedded systems C has become a standard feature amongst most microprocessors, whereas C++ is not so common. An article of Rosenthal has indicated that there is an alternative to a C++ compiler, a preprocessor. "If you want to use the language for a processor that doesn't have a native C++ compiler, try using a preprocessor that translates a C++ program into standard C code" (Rosenthal, 1992, p. 3). This option could be a substitute for a compiler however further on Rosenthal states: "A more important issue is how to debug the code. Specifically, how will a source-level debugger work with C code that you didn't write and that you don't want to patch." (Rosenthal, 1992, p. 3). With the potential difficulty of debugging the preprocessor option was not considered.

Java requiring a virtual machine means the code is easily transportable between different microprocessor platforms. There are three main Java platforms:

- J2SE, Java 2 standard edition.
- J2ME, Java 2 micro edition.

- J2EE, Java 2 enterprise edition.

J2SE is the java platform that is typically used for PC based applications, J2ME contains a subset of J2SE with some additional features to make it suitable for embedded applications. J2ME has been designed to use the KVM virtual machine, where KVM stands for either 'Kaffe Virtual Machine' or 'Kilobyte Virtual Machine'. The KVM has been designed and optimised for small microprocessor based applications. Whereas the JVM which stands for 'Java Virtual Machine' and is typically used for J2SE and J2EE which are designed for use with PC based applications that do not have the resource limitations of a small microprocessor. J2EE is the largest of the Java platforms and is understood to include libraries for network server applications.

As explained in the section entitled 'The Structure of Java and J2ME' the same compiler that is used for the PC can be used on embedded platforms. Java 2 Micro edition (J2ME) is the Java environment designed for microprocessor based systems. However a Java virtual machine is required for the target platform. As explained in the section Java 2 Platform, Micro Edition there is freely available C source code for a virtual machine known as the KVM. This means that where there is a C compiler available, the KVM can be ported to this platform providing there are sufficient resources. For this reason using Java would still require a C compiler.

One of the big considerations of choosing languages, was compatibility of source code with the developer's communication software that is written in C. Considering the size of the communications software, it was not an economical option to port this software to another language. This meant that the language chosen for the IEC 61499 run-time software would have to be compatible with C. As C++ is an extension of C, the existing C communications software would be compatible with C++. Java and C are only partially compatible, as described below. As Java does not have any I/O capability, there is provision for a native language interface. "The Java Native Interface (JNI) is a standard cross-platform programming interface included in the JDK. It enables the

programmer to write Java programs that can operate with applications and libraries written in other programming languages, such as C and C++. Using JNI, Java native methods can be written to access an existing library in C++ and make it accessible to Java code." (Wei, 2002, p. 49-50). Wei mentions the use of C++ as the native language, however Liron states that other languages can also be used: "If required, you can develop your JNI library in languages other than C/C++, although you will have to do your own translation of entry points and arguments." (Liron, 1999, p. 6). From the above information it seems that using JNI, a Java software routine can call subroutines from other languages. With the JNI facility, it indicates the existing communications software of the developer's can be used with Java.

The object orientated structure of the IEC 61499 function block programming language has function blocks operating independently of each other. This showed that multi-threading was a requirement. Multi-threading is multitasking of different objects, components, or tasks that require servicing independent of the main program execution cycle within an application. Both Java and C++ were investigated for their capabilities of multi-threading. An article comparing Java with C++ multi-threading illustrates the differences. "One of the features of the Java language is the native multi-threading support. The Java thread support is inherent in the language rather than added on as libraries as for C and C++." (Howard, 1997, p. 1). C++ only has multi-threading available if the operating system supports it. The development platform used by the author did not have any multi-threading support because it does not have an operating system. To build multi-threading support, either developing this functionality, or porting this functionality from another operating system would be a potentially difficult task.

Klander describes how multi-threading differs between operating systems: "Threads and synchronisation objects are implemented on Win32 platforms like Windows 95 and Windows NT, but they are not available on Win32s. If you are planning a Win32s application, you won't be able to use threads or synchronisation objects in your

application." (Klander, 2000, p. 275). This article implies that multi-threading is part of the operating system. With the developer's embedded platform there is no operating system. To use C++ with multi-threading an operating system would be required. At this stage the developer does not intend to use an operating system.

It has been noticed that with C, care is required with management of dynamically allocated memory, as cleaning memory areas that are no longer allocated is a manual process. This requires manually freeing memory and defragmenting unused memory locations. "It (Java) also adds new features that are not available in C++, most notably: Automatic garbage collection simplifies dynamic memory management and eliminates memory leaks." (Barr, 2002, p. 1-2). The virtual machine of Java handles memory management issues that have to be manually handled with C. This reduces an area that can cause software bugs that are hard to find, however is a cost in speed. This is one of the reasons why Java is slow.

In discussions with the developer it was decided to use Java despite the fact that it uses more microprocessor resources than C++, for the following reasons:

- Compatibility of code between different product manufacturers and researchers. This allows more support with others performing function block research.
- It has automatic garbage collection for dynamically allocated memory.
- It has multi-threading capabilities.

Both automatic garbage collection and multi-threading are a requirement for developing an IEC 61499 run-time environment. Using a language that supports these features will automatically simplify development:

- C++ compilers are not as readily available. For Java, a standard compiler is used for all platforms, but a virtual machine is required. Although a virtual machine is not available for all platforms, as discussed later with the J2ME,

there is C source code available for an embedded virtual machine known as the KVM.

- Java is portable. Compiled Java can be transported between platforms without porting. (Note: all native functions will need porting).
- When developing function block definitions the FBDK software generates a Java template. This is a feature that could save a lot of time.
- Java can not interface directly to I/O, but there is provision to access C software routines that can interface to I/O.
- Java has been used by most of the other researchers developing function block software.
- The developer prefers to follow the same path that other companies are taking with their function block software development. Other non-technical considerations made by the developer company:
- Java offered the quickest time to get a product to market. This is a cost saving in labour and allows launching of a product before the competitor.
- Joint development efforts with other companies and tertiary research institutions. With contractual agreements to protect each company's potential markets.
- Marketing point of view as a sales pitch to say that we support Java.

With the above points and the fact that it is technically feasible to use Java the decision was made even though it had technical disadvantages. Which are:

- Larger memory and processor speed requirements.
- I/O access not as efficient.
- The requirement to port a Java virtual machine.

## Java 2 Platform, Micro Edition

The task of finding out what is available with embedded Java was not in the scope of this project. One of the developer's staff members investigated the availability of Java

tools on embedded platforms. The results of his investigation are to use Java 2 Micro Edition CLDC. "Embedded Java is taking the Java language and streamlining it to allow the code to be executed in a processor. This is no mean feat as Java is a relatively resource hungry (CPU cycles, memory) language. Sun (Java's creator) achieved this by creating the J2ME software development kit. This stands for Java 2 Micro Edition. This includes cut down facets of the language and comes in two flavours. These are: Connected Limited Device Configuration (CLDC) and the Connected Device Configuration (CDC) and Foundation Profile (FP)." (Edgar, 2003, p. 2). The configuration chosen for the project was CLDC, this decision being made by Edgar. The CLDC configuration includes the following I/O:

- Network communications.
- Real-time clock interface.
- RS232 communications.
- Java native language interface. This allows interfacing to another language such as C.
- Uses a cut down version of the Java virtual machine.

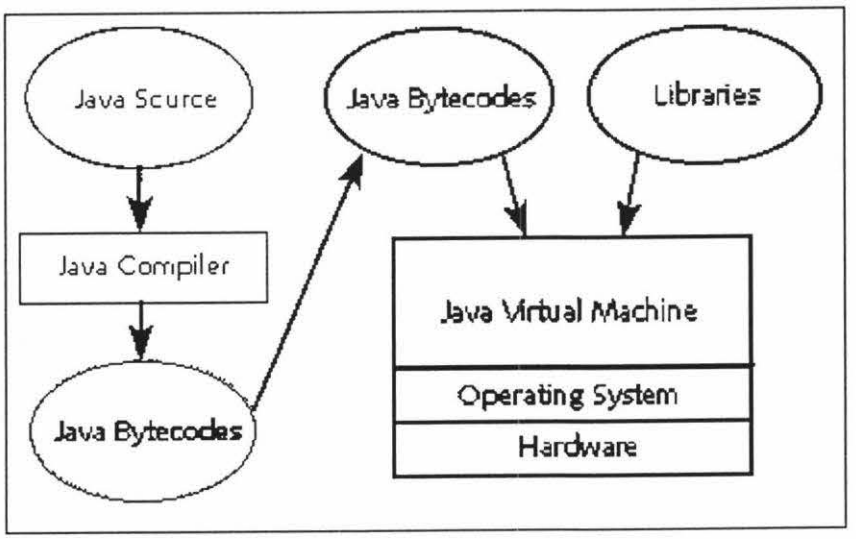
The CDC configuration includes everything of CLDC with the following extra features:

- Serialisation, allowing objects to be saved to files.
- Network security features.
- Uses a full Java virtual machine (not cut down).

## The Structure of Java and J2ME

The diagram below illustrates the structure of Java

**Figure 3.2 The architecture of Java**




(Barr, 2002, p. 2)

It shows Java as a cross between a compiled language and an interpreted language. The Java source code is compiled into a byte code which is then interpreted by a Virtual Machine (VM). The Java virtual machine is an application that interprets the Java byte code. The reason for Java using this method is it allows code portability between any microprocessor platform that has a Java virtual machine. The drawback of this approach is a reduction in speed and use of more memory due to the virtual machine overhead. The byte code format is standard between Java platforms. With Java there are standard function libraries for maths, and other program manipulation functions. With J2ME a lot of the library functions have been stripped down to the basics, and the virtual machine known as KVM has been written in C, so it can be compiled on any platform which supports an ANSI C compiler. The same Java compiler that is used for the PC is used for compiling code for embedded devices, with the exception the use of a pre-verifier. Pre-verification is performed after compilation to check that the compiled code does not include any unsupported features of the target platform, and modifies the structure of the compiled code, as the KVM is strict on the format of the compiled code. For reduction of space with an embedded platform some of the features like floating point arithmetic can be removed from the KVM, as well as some of the library

functions. If the pre-verifier has successfully checked all the code, then the Java software can be used with the KVM.

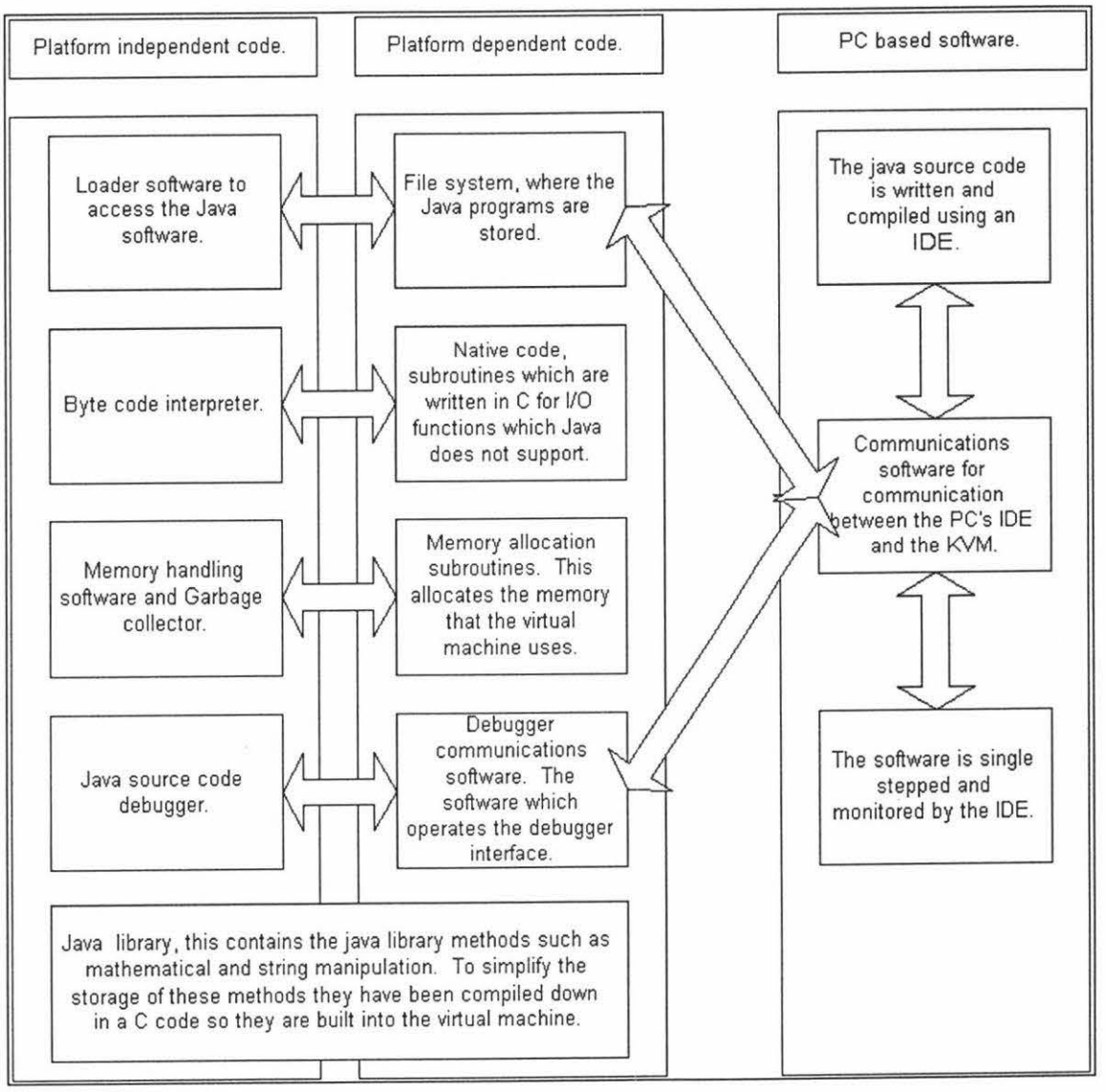
## Porting of the KVM to an Embedded Platform

The KVM source code which has been ported for the PC Microsoft win32 platform is available from the Sun Java web site www.sun.com. The source code for the KVM was divided into two sections: platform independent code, and platform dependant code. The platform independent code consists of the virtual machine core. This code should compile on any ANSI C compiler, which it did, apart from a few variable definitions that needed changing. The platform dependant code needs porting for the target platform. The areas where customisation was required are:

- The execution of the virtual machine and its software applications, this refers to the boot up sequence of the microprocessor.
- Loading of Java software onto the embedded platform.
- A file system with a directory structure to store the Java class files. (The byte code is compiled into class files).
- Input/output routines.
- The software debugging interface.

The figure below illustrates the structure of the software tools and Java virtual machine.

**Figure 3.3 Simplified Diagram of the Embedded Java KVM**




It shows two sections of code; Platform independent code and platform dependent code. The platform independent code requires compiling to the target platform, but it is ANSI C compliant and does not require any I/O except via the platform dependent code. In theory the platform independent code will not require any changes to port to a different platform. The platform dependent code is called from the platform independent code to access platform specific I/O routines. This software will require changes when porting to a different platform. Inside the virtual machine's source code are the following sections:

- Loader software to access the Java software, this is collection of subroutines which extract the Java class files (complied java code) from a file system or ROM.
- The byte code interpreter is micro-code that executes the byte code. This accesses the native code subroutines that are typically written in C when I/O functions need accessing.
- Memory handling and garbage collection. This handles creating and deleting of members and objects. Garbage collection automatically deletes objects and members that are no longer required as well as defragmenting the memory. The platform dependant code is used for allocating the memory that is required for the Java heap.
- Java source code debugger. The Java source code debugger is a group of subroutines that allow the software developer to insert break points and watches into the Java program. This talks to platform dependant code that communicates with a communication medium to the PC development tools.

The PC based software is used to write the Java program, download to the device and communicate with the debugger software on the target if required. The Java library comprises of the methods (subroutines) in Java that are part of the J2ME platform. This code sits between platform dependent and independent code because methods may be added or removed from it depending on what is required for that platform.

The byte code interpreter is the code that converts the Java byte code to the machine code of the target microprocessor. IDE stands for "Integrated Development Environment" which is the Java development tool that is run on the PC. Software is required on the PC for downloading the Java programs and the communication interface for the debugger. A debugger is software that allows single stepping of the Java source code.

The first stage of porting the KVM virtual machine was to modify it to run on the target platform with a simple Java program. The program chosen displays a simple message on a console device that says 'Hello World.' The console for the embedded system was a PC acting as a virtual terminal connected via a serial port.

## Platform Independent Code

The initial stage of porting the KVM was to get the platform independent source code to compile. As this code is designed to compile on any ANSI C compiler, this task should be simple. However when compilation was tried it was more difficult than expected. Most of the problems that occurred resulted from differences between compilers. "In order to be able to compile the KVM code base, you must have a C compiler capable of compiling ANSI-compliant C files." (Sun Microsystems, 2002b, p. 5).

The source code for the KVM was written in ANSI-compliant C files which means that any C compiler that is ANSI C compliant should compile the code. When compiling the code it was observed that this was not the case. It was noted that the developers of the KVM package made sure that the code was ANSI C compliant, however there were still inconstancies between the two platforms. To locate these problems the C compiler was run and modifications made to the code until it compiled. The code was tested using an "In Circuit CPU Emulator" which can single step the C code. Parts of the C code were single stepped, and from observing the results of each step, the problems could be located.

Most of the problems were related to the definition of software variables. They were as follows:

- The compiler supports a maximum of 255 bytes of local variables per subroutine. Making some of the variables global solved this.
- Conflicts between 16 and 32 bit pointers: Pointers are software variables that point to locations in memory. There were problems with the 16 bit pointers

needing to point to 32 bit memory locations. This was solved by making all pointers 32 bit.

- Incorrect variable definitions: Depending on the definition of the variable, the amount of memory that a variable uses can change. In the platform independent code, some of the variables were not allocated enough memory. Changing variable definitions and casting variables as different types where appropriate solved this. Casting a variable is an instruction to tell the compiler that this variable is implemented as a different type while that particular line of source code is being executed.

For all of the above problems the corrected code was done generically. This means the code should be transportable to a different platform without having this problem again.

## Platform Dependent Code

When the KVM source code was first received, it was ported to run on the PC win32 platform. For this code to run on the embedded platform, modifications were required. The sections of source code that needed modification are:

- Run-time functions.
- Storage of Java programs (Class files).
- The J2ME_CLDC Java library. This is the library of methods, (procedures, functions, or sub-routines are called methods in object orientated languages) for example mathematical and string manipulation methods.
- Debugger communication interface subroutines.
- Native code for the J2ME_CLDC Java library. These are the C subroutines that are used to interface the J2ME_CLDC Java library to the I/O ports on the embedded platform.

### Run-time Related Functions

The run-time related functions are subroutines that the KVM calls for the following:

- Initialisation of memory.

- Communicating fatal error messages to the software developer or user.
- Finding out the system time.
- Execution of the virtual machine.

The details of what is involved are discussed in the next sections.

## Initialising Memory

The KVM uses a section of memory called the heap for dynamic allocation of members and objects (variables are called members in object orientated languages). The dynamic memory allocation and deallocation is managed by the KVM's garbage collector. The garbage collector checks the KVM's heap to clean up and delete members and objects that are no longer relevant. The KVM's requirement for its heap is one contiguous block of memory, which is no less than 16 kilobytes, and the size is a multiple of four. The start memory location of the heap must be a memory location with its address as a multiple of four. Overcoming the problem of the heap having to start with a multiple of four was simple. An extra four bytes were allocated and the next address which is a multiple of four was used. Adding an extra four bytes on the end of the heap allowed for the worst case scenario of having to add another three bytes onto the start address.

There are two methods of allocating this heap memory on the target platform. The initial method was to allocate a section of the C language's heap area that is reserved for dynamic memory allocation. This was chosen initially because the KVM that was ported to run on the PC win32 platform uses this method. At a later stage in the porting of the KVM, the method was changed to allocate the heap to an array in the C source code. The reason for changing the approach was dynamically creating the KVM's heap was considered unnecessary, as typically with the embedded application the KVM will be allocated a fixed amount of memory. Another reason for this is that it is easier to keep track of the memory usage when developing if the heap is given a fixed amount.

When the Java heap size was extended beyond 32 kilobytes, the Java virtual machine ceased working. When running the CPU emulator it was observed that some of the

memory allocation variables were 15 bit (16 bit two's complement). This resulted in the virtual machine failing when the memory size was greater the 32 kilobytes. Changing the memory allocation variables to 32 bit solved this problem.

## Handling of KVM Fatal Errors

When the KVM has a fatal error which is typically related to the KVM not functioning correctly, lack of memory, or corrupted Java application byte code, the KVM will transmit an error message. With the Java application running on the PC platform the KVM errors are displayed on the screen. On the embedded platform these error messages are sent out of the serial port.

## Calculation of the Current System Time

The KVM porting documentation stated that the timing related subroutine required was to "return the time, in milliseconds, since January 1, 1970 UTC" (Sun Microsystems 2002b, p. 29). The developer's target platform does not have a real-time clock. This is a clock that runs in the background keeping track of the current date and time. Initially it was thought that this timing subroutine was only for use with the calender related methods in Java, so whenever this subroutine was called a zero was returned. However it was later discovered that all timing related functions use this subroutine, so it was modified to return the time in milliseconds since the microprocessor was powered up. For proof of concept this was sufficient, as global synchronisation was not required for any of the applications used in this project. If synchronisation is required across multiple nodes on a network the function block specification has timer related function blocks within each node. To synchronise across multiple nodes, multicast network communication could transmit a strobe pulse as a command to synchronise the individual timers in each node. This method of synchronisation would require a master synchronisation device, and could be the device that has the highest priority on the network or the device that is supervisor for that particular process. However this introduces a single point of failure into the system, which is what distributed systems are designed to avoid. A possible way of overcoming this is to have prioritised backup

devices for synchronisation. For example if the master device does not send its synchronisation pulse within a time frame another device can assume that the master device is off line or faulty, and become the master.

## Execution of the Virtual Machine

Starting of the virtual machine on the win32 platform and the SNAP Java prototype development board is done by a command prompt or a Windows shortcut. For the production platform, the execution of the Java virtual machine will be performed during microprocessor boot up. However, for Java software development on an embedded platform, having the Java virtual machine start on boot up is not satisfactory. Before it is started the user may need to do the following:

- Download the Java program into the target platform.
- Select the operation mode required to run the Java virtual machine, which can be: run with or without debugger support.
- Select the location of the class file that has the main method in the Java program.

When developing Java software on a PC, the software developer may use an IDE (Integrated Development Environment) software package. The user interface for this is often simple and quick to use and is recommended for embedded Java development. Most of the IDE software packages available are designed for use with the PC platform. There are IDE packages available for embedded Java but from investigation they use cell phone platform simulators, instead of interfacing directly to the target platform. The easiest way to establish a link between the target and host was to make a simple serial communications protocol program. This would transfer the Java program to the target platform and select the mode in which the Java virtual machine is to run. The serial communications protocol was later changed to ethernet. The reasons for this are explained in the section: Debugger Protocol Subroutines. To use the serial interface the software developer is required to make batch files with a list of the Java files that need downloading, and the options required for execution of the Java virtual machine. While this arrangement is not ideal, it was the simplest option without modifying the source

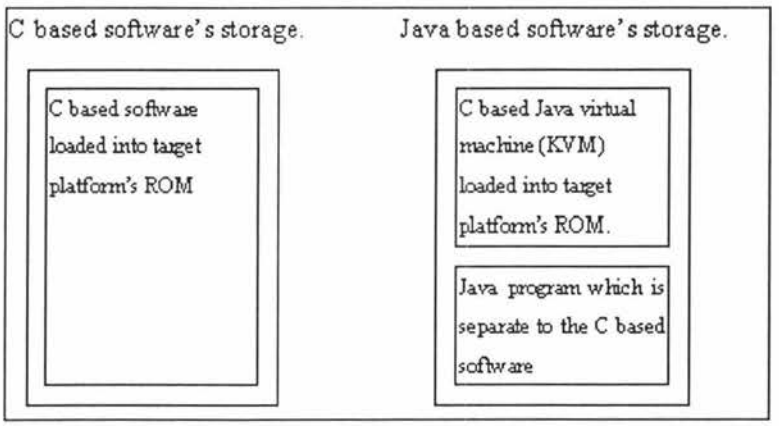
code of an IDE package. Modifying an IDE package would require the package's source code, which is usually not released by the software manufacturer.

The serial communications protocol consisted of the length of the data packet sent, followed by a command, file name, and if downloading the program, the file data. There was no error checking and packet retries, as at this stage the idea was to keep it basic as serial communications was only an interim step before the interface was changed to ethernet. Apart from slow download times the serial communication protocol was effective.

## Storage of Java Programs (Class and Jar files)

All software in the developer's product range to date has been developed in C. The C compiler compiles the C code into a binary data format, which is then burnt into the ROM of the target processor platform. This means that the compiled program is in a storage structure that has been designed for that microprocessor. With the KVM, the microprocessor platform starts off with the virtual machine loaded into the target platform as described above. The Java application is then loaded as a separate package into the target platform. The figure below explains this structure:

**Figure 3.4 Comparison between storage structures**




Java programs are stored on the PC with each class stored as a separate file, unless the individual class files are compressed into a Jar file. When the Java virtual machine

accesses the code in a class, it will open that class file, unlike C based software, where the code is stored in one block. When looking at the source code for the KVM, it was noticed that the same Java program storage structure was used. This resulted in the need for a file system.

After investigating file systems, such as FAT16, it was decided to keep the file system for the embedded platform simple, reducing software overheads. FAT 16 is a file system used on pre Windows 95B PC's. "FAT is a very simple file system which is nothing more that a singular linked list of clusters" (Mega-Tokyo, 2003, p. 1). The file information is stored in a table format that is shown below.

**Table 3.1 File information table**

| Memory location | Field Length (bytes) | Field |
|---|---|---|
| 00h | 8 | Filename padded with spaces if required. |
| 08h | 3 | Filename extension with spaces if required. |
| 0Bh | 1 | File attribute byte. |
| 0Ch | 10 | Reserved or extra data. |
| 16h | 2 | Time of last write to file (last modified or when created). |
| 18h | 2 | Date of last write to file (last modified or when created). |
| 1Ah | 2 | Starting cluster. |
| 1Ch | 4 | File size (set to zero if a directory). |

(Fox, 2002, p. 6)

When investigating the requirements of a file system it was decided that using an existing file system such as FAT16 was unnecessary and included extra overheads such as file fragmentation. Fragmentation occurs when a file can be broken into sections that are stored in different areas of memory. It is commonly used with PC hard drives to allow files to be deleted and new files written without needing to reshuffle data, to avoid wasting areas of disc memory. The need for features such as fragmentation is unnecessary, as when a Java program is downloaded to the target platform it will be writing into dynamic memory where there is no old data retained.

This resulted in development of a file system that contains a file information table as shown in table 3.2. The last item in the table is the location of where the next file was stored. The Java class file data was simply copied from the PC where it was compiled and placed into an area of the target platform's memory.

**Table 3.2 File information table used for embedded Java**

| Memory location | Field Length (bytes) | Field |
|---|---|---|
| 00h | 4 | Pointer to the memory location that the file name is stored in. |
| 04h | 4 | Size in bytes of the file. |
| 08h | 4 | Pointer to the start of the file. |
| 0Ch | 4 | Pointer to the start of the file. |
| 10h | 4 | Pointer to the next file in the linked list. |

Initially for the development platform the data was stored in RAM with the exception of library functions. These were stored in ROM to simplify downloading of software. During the development of the IEC 61499 run-time environment memory became scarce, so the class files where relocated to a serial flash chip using the same file system structure. This provided an adequate file system for the KVM to use. The simple file system performed as it was required. Although it did not support deletion or modification of files, other than deleting all files and restarting, it successfully supported Java and demonstrated the use of function blocks. The limited features later proved advantageous when memory usage became scarce.

To get the KVM running initially, the Java programs were downloaded as individual class files. When the developer received the third party IEC 61499 run-time environment, there was an attempt to use the Jar file format. The reason for using the Jar file format is that the class files are compressed into a single file. The KVM source code includes a decompressor. This source code was relatively easy to port, however some problems arose the with definition of variables. The C compiler for the original platform that the KVM was ported to compile all fixed point variables as 32-bits in

length regardless of the variable definition. The embedded target platform compiler is more efficient and compiles the code to use the correct number of bits for that variable definition. This meant some of the variables were incorrectly defined. There was an attempt to debug the decompressor, however the task became more involved than originally anticipated.

While trying to debug the decompressor, it was noted that for a production version of the target platform, embedding the IEC 61499 run-time software into the virtual machine was a more viable option. The IEC 61499 run-time environment is a Java application used for the distributed control and will be discussed later. Reasons for embedding this is, the space that is used in ROM appears to be less, and the loading time considerably quicker. For instance, during a test run, embedding the run-time into the KVM resulted in immediate boot up of the run-time, whereas it normally takes 20-30 seconds. The other reason was that during development of the IEC 61499 run-time environment it was observed that the developer's target platform does not have enough speed and memory resources for a production run. However it is capable of proving concept. As a solution to overcome the lack of memory problems, the Java class files were stored in a 2Mbyte serial flash chip that has enough space to store the full IEC 61499 run-time environment. Thus debugging the Jar file decompressor was abandoned.

## The J2ME_CLDC Library

The J2ME_CLDC library contains the basic methods that the Java software developer uses. There are two ways of storing these library files: download the individual class files using the file system described above, or use JavaCodeCompact (JCC). "At the implementation level, the JavaCodeCompact utility combines Java class files and produces a C file that can be compiled and linked with the Java virtual machine. JavaCodeCompact provides an alternative means of program linking and symbol resolution, one that provides a less-flexible model of program building, but which helps reduce the VM's bandwidth and memory requirements" (Sun Microsystems, 2002b, p.

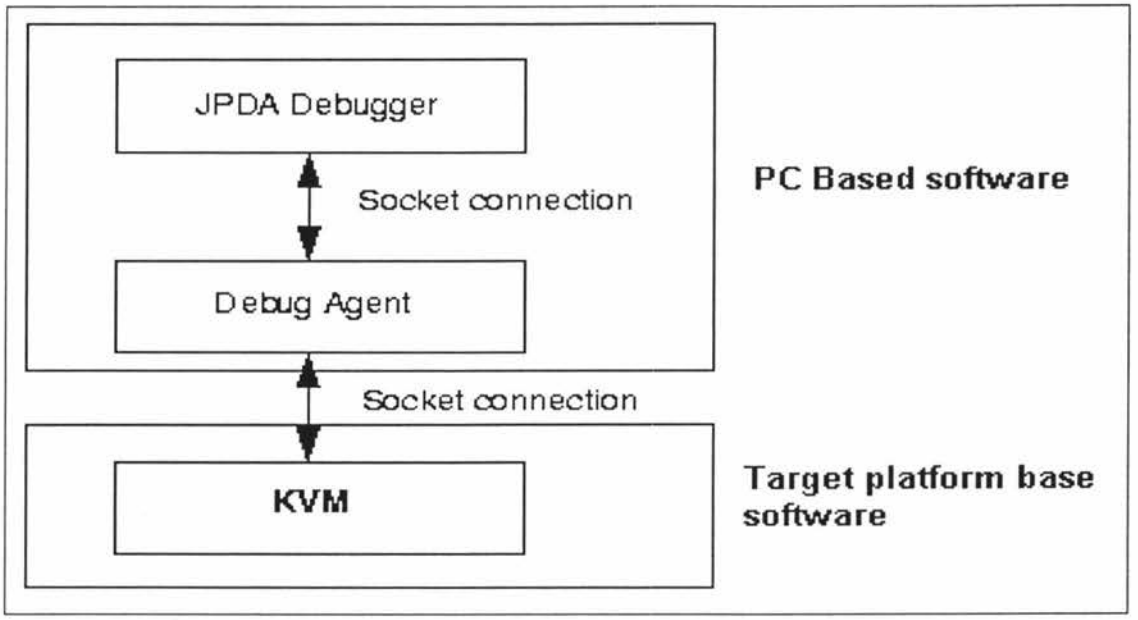
75). JavaCodeCompact was the chosen option as it allows for all the class files from the standard library to be combined into the C source code of the virtual machine. This offers simplicity in compiling the virtual machine and it also means the standard Java library class files do not need to be downloaded every time the application software is modified. The disadvantage of JavaCodeCompact is that there is more work involved in modifying the standard Java library.

There are instructions in the KVM porting guide (Sun Microsystems, 2002b, p. 75-79) on how to compile the Java source files into class files, which are then converted into C files. Apart from removing the ethernet functionality from the Java library there were no problems with compiling the standard Java library. The ethernet functionality was removed, as the developer's target platform does not support ethernet.

## Debugger Protocol Subroutines

A debugger is a software package that runs on the PC allowing single stepping of Java source code and inspection of members. This allows the software developer to observe the program operation and find software bugs. The KVM has optional debugger functionality that can be switched on before compiling the virtual machine. This option was chosen because tracing program flow can save development time, however the KVM was only run in this mode while debugging the software. The communication between the IDE software's debugger and the KVM is termed the KDWP (KVM Debug Wire Protocol). "KDWP was designed to be a strict subset of the JDWP (JDWP is the protocol used between a debugger and a virtual machine on the PC), primarily based on the resource constraints imposed on small devices. In order to make the KVM run with a JDPA-compatible (JDPA is a debugger communication standard) debugger IDE without a huge memory overhead, a Debug Agent (also known as a debug proxy) program is interposed between the KVM and the JPDA-compatible debugger" (Sun Microsystems, 2002a, p. 2). The figure below shows how the JPDA debugger (this is part of the IDE software package), Debug Agent and KVM are connected to each other.

**Figure 3.5 Java Debugger Interface Architecture**




(Sun Microsystems, 2002a, p. 2)

From the diagrams shown in the KVM porting guide 'Socket connections' are used as the interface between the three software packages. "Socket connections" is a standard way of connecting to TCP/IP network protocol stacks. The connection between the JDPA debugger and the debug agent is typically an internal local host connection. The connection between the KVM and the debug agent is through ethernet. When initially porting the KVM, the developer's target platform had no ethernet functionality. To get around this problem the KVM and the debug agent where modified to use RS-232 serial communications as a substitute. RS-232 is a serial communications standard that most PC's support. Converting the software to use RS-232 serial communications was straightforward. The only problem was the byte order of the software variables. This was different between the communications protocol and the target microprocessor. Although the Java debugger started working, it was unreliable. It became apparent that simply sending the KDWP packets out of a serial port was going to mean the serial communication protocol was too simple. The idea of using checksums and packet retries was attempted, however this task became complicated and another problem arose that is discussed in the next paragraph.

When running the debugger software it was noticed that there was approximately a five to ten second delay on starting up and initialising, and then approximately a one second delay when stepping through the program. Initially the assumption was made that the serial connection with a speed of 57600bps was too slow, compared with the possible 10Mbps ethernet connection. As an attempt to find out what was causing the data corruption with the debugger interface, the bit rate was temporally dropped to 9600bps. The expected result was for the debugger to run more reliably, with expected speed reduction. The result was increased reliability and, for normal single stepping the source code, no noticeable speed reduction. However initialisation of the debugger took longer. What was also noted was that the same serial link was used to download the Java class files to the target platform at 57600bps, however this had not caused any problems. Comparing the two different data transfers, most of the data is transferred from the PC to the target platform for class file downloading, whereas for debugging, most of the data transfer is from the target platform to the PC. This observation indicated that there was a problem with the PC Java serial communications API. The serial communications API is a software library that contains methods to communicate with the serial ports. A further observation showed that the introduction of wait states in the KVM serial communications routines improved reliability. From the above findings it was concluded that the KVM does support a debugger interface which offers the facilities that are expected by a software developer, but RS-232 serial communication is not suitable. Work from Sarah Browne's research into embedded ethernet (Sarah Browne, personal communication, 2003) showed that there are serial to ethernet communications adaptors available. Using one of these adaptors, the serial communications protocol inside the virtual machine of the target platform required only minor modifications, and the original ethernet KVM debug agent could be used. The results of using the ethernet to serial communications adaptor confirmed the belief that the PC Java serial communications API is unreliable. The debugging interface became reliable, and the speed improved considerably.

## Native Code for the J2ME CLDC Software Library

The Java language does not support any I/O methods. To solve this problem there is provision to build native methods that are written in other languages such as C. These native methods provide the functionality of communicating with I/O devices. In order to get a simple 'Hello World' application operational, I/O communications are required to output the 'Hello World' message to the display. Coincidently the output methods in the J2ME CLDC where configured to send the data out of the serial port which was planned for use. This meant that for the simple 'Hello World' application no modifications were required.

However to have full functionality of the J2ME CLDC library the following needed modifying:

- Serial communications protocol.
- TCP/IP socket interface. (Sockets are a standard interface used for software programmers to interface with TCP/IP communication stacks).
- The KNI (KVM native interface).

The serial communications protocol was relatively simple to modify and the native code routines were changed to suit the target platform. The biggest problem was finding out how to use the J2ME CLDC serial communications API. This was solved using the Java debugger interface and single stepping through the Java source code to find out what commands the API requires. The TCP/IP socket interface at this stage did not have any modifications made to it, as the target platform does not support TCP/IP or ethernet.

The KNI (KVM Native interface) is the interface for Java programs to call native language subroutines, such as C and assembler. The KNI interface allows the programmer to embed native code into the KVM. Originally the KVM source code was set up so the Java class files, which accessed the native code had to be embedded into the KVM. For development this procedure requires a number of steps and is time

consuming. To simplify JNI for software development the KNI interface was modified so the Java code did not need embedding into the virtual machine. The custom API's for the target platform are, for example the DeviceNet interface which is discussed later in chapter 7.

## Summary

From the discussion above and tests conducted that are discussed in chapter 4 it was concluded that Java is a suitable language for development of an IEC 61499 run-time environment for use on an embedded platform that has a powerful microprocessor.

# Chapter 4 Evaluation of Embedded Java

The purpose of this evaluation test is to compare the performance of embedded Java applications with embedded C. Two applications were tested because there were deficiencies in the first test application.

The product used for the first test was a bar code scanner interface module the TCS-DNSI-SICK. This is a protocol converter for a bar code scanner to be interfaced with a PLC (Refer to appendix B for the product data sheet).

The reasons for performing this test are:

- To establish if the bar code scanner application could be written in Java.
- To compare memory usage with C and Java.
- To compare the code efficiency between the two languages in execution speed between two applications that perform the same functionality.

The second test application is discussed in the section on Bench Mark Testing. This test application was performed because the results of the bar code scanner application showed deficiencies in the test design.

## Test Procedure

### Memory Usage

The memory usage of the two applications is compared in the following areas:

- Essential software libraries required by both applications. Example, DeviceNet communication software and platform hardware related subroutines.
- Memory required for the additional Java virtual machine.
- The code required by the application itself.

Analysing the memory map files generated by the C compiler and the file size of the Java application evaluates the memory usage.

## Processor Speed Requirements

This test compares the time it takes for each application to execute. Both applications use similar microprocessors. The C application used a Mitsubishi M16, and the Java application used a M16 CPU emulator. The Java application used the CPU emulator because the standard M16 did not have enough memory at the time the test was performed. The Java virtual machine was configured without the debugger functionality for these tests. When it was performed it was assumed that both microprocessors ran at the same speed. This was never checked and was not considered until months later. This is further discussed in the results section. The test set up is shown in the following figures.

**Figure 4.1 Diagram of test set up**

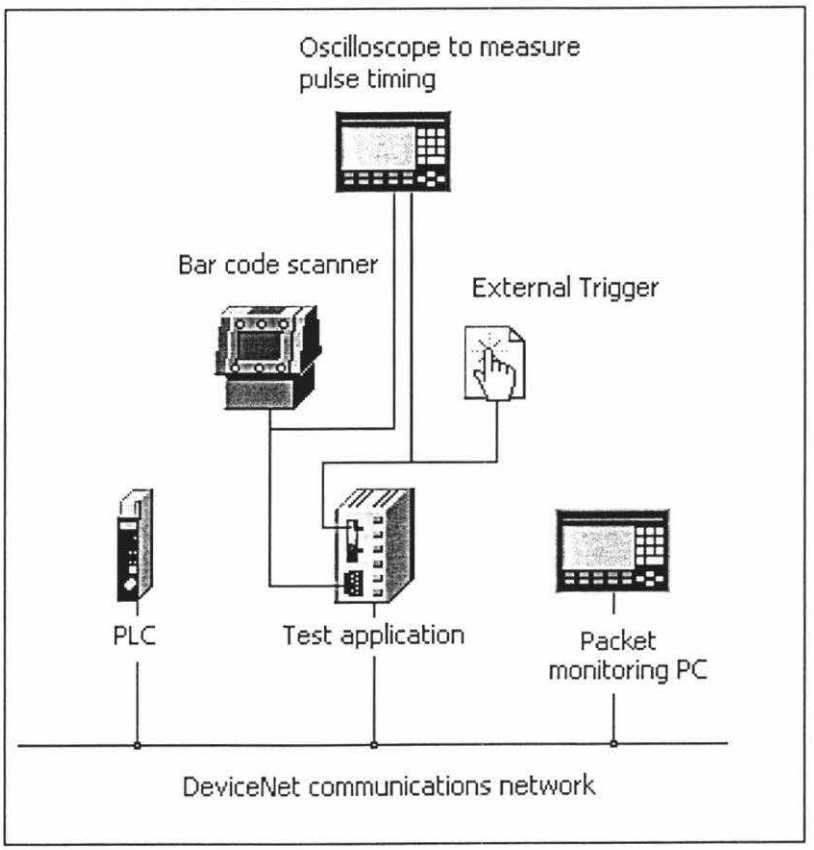
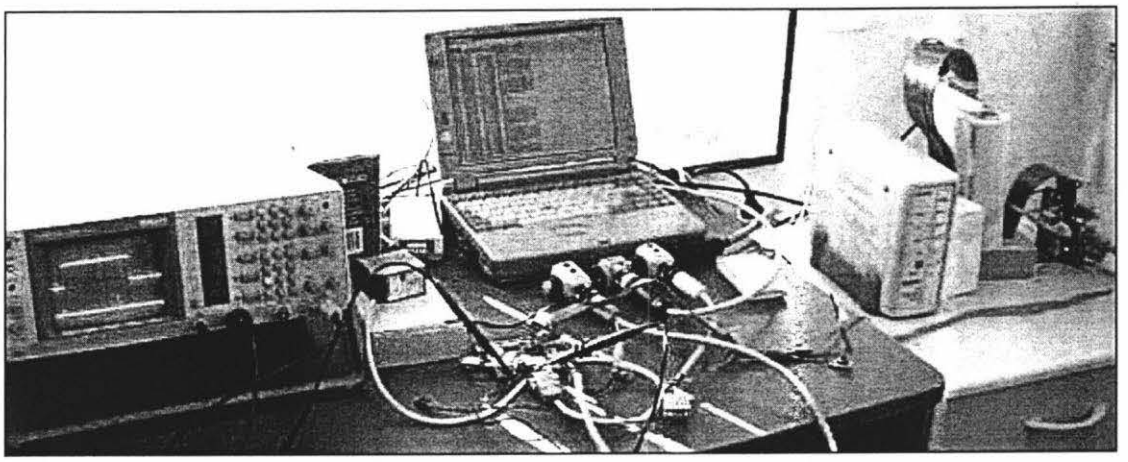


**Figure 4.2 Photograph of the test set up**



The diagram above shows a network of devices using DeviceNet which is situated in front of the laptop. A PLC is acting as the central controller. This is not shown in the photograph as it is mounted on a wall in a different room to the test. The test application device has a bar code scanner attached to it by a RS-232 serial link. This is located on a box between the laptop and the oscilloscope. The bar scanner is interfaced to the module that is the in-circuit microprocessor emulator that is located on the right of the photograph. Two tests were done; one using an external trigger to trigger a bar code scanner to read, where an oscilloscope is used to measure the trigger response time. The second measurement used a 'Packet Monitoring PC' (the laptop) to measure the time delays between a PLC trigger and the bar code scanner read. The following were tested:

- The time delays between when the PLC requests a bar code read and when the PLC receives the response. This is measuring the time from receiving a DeviceNet message, processing it, transmitting it to the bar code scanner, the bar code scanner reading bar code, the bar code scanner transmitting the bar code to module, the module receiving the bar code, processing the message, and then transmitting it over DeviceNet to the PLC.
- The time delay between the PLC's bar code received acknowledgment message and when the bar code scanner interface has received this message. This is measuring the time delay from receiving a DeviceNet message,

processing it and then transmitting the acknowledgement message back on DeviceNet.

## Results

The results of the memory usage for the two applications are as follows:

**Table 4.1 Memory usage between C and Java applications**

| Type of Memory | C application (bytes) | Java application (bytes) | Essential Java libraries (bytes) | Java virtual machine (bytes) |
|---|---|---|---|---|
| RAM | 167 | 65536* | 4753 | 45994 |
| Constants, (ROM) | 1140 | | 1177 | 137168 |
| Code (ROM) | 2194 | 39295 | 54701 | 256558 |

* The amount of RAM for the Java application could not be measured, 65536 bytes was the amount of RAM that the virtual machine could access for the application.

The above table shows that the Java application uses vastly more memory. The code usage of the application itself has increased 18 times. This ratio could be improved to approximately 11 if the Java virtual machine was modified to use the Jar file storage format. The RAM usage is hard to estimate, as the Java compiler does not give a memory usage output after compiling. During this test the virtual machine had 65536 bytes available to run the program. In the early stages of development the Java virtual machine had 32768 bytes available, which was not enough. The addition of the virtual machine itself uses approximately 500kbytes of ROM, which is an overhead before the Java application is loaded. The raw data from these tests are shown in appendix C.

The timing measurements are documented in the table below. All response times are in milliseconds. A sample size of 10 tests was used.

**Table 4.2 Response times between Java and C application**

| Test number | Java application bar code response | Java application acknowledge response | Java application trigger delay | C application bar code response | C application acknowledge response | C application trigger delay |
|---|---|---|---|---|---|---|
| 1 | 266 | 145 | 400 | 113 | 8 | 96 |
| 2 | 358 | 151 | 192 | 139 | 7 | 105 |
| 3 | 379 | 138 | 304 | 209 | 12 | 101 |
| 4 | 438 | 230 | 182 | 210 | 13 | 103 |
| 5 | 519 | 193 | 229 | 210 | 11 | 102 |
| 6 | 727 | 159 | 209 | 208 | 14 | 104 |
| 7 | 742 | 461 | 55 | 211 | 13 | 102 |
| 8 | 721 | 256 | 206 | 213 | 9 | 104 |
| 9 | 590 | 259 | 314 | 216 | 16 | 102 |
| 10 | 300 | 62 | 121 | 214 | 15 | 99 |
| Average time | 504 | 205.4 | 221.2 | 194.3 | 11.8 | 101.8 |

The above table indicates that on average the Java application is 2.5 times slower than the C application for the bar code and trigger response, while for the acknowledgment response it is 17 times slower. It was also noticed that the Java application has a wider range of time delays compared to that of the C application. This shows that the Java application is not as predictable in execution time as the C application.

The large difference between the two tests indicated that the tests overlooked a number of issues. When the time differences were calculated, no consideration was taken for the time it takes for the DeviceNet communications to process a message. The DeviceNet software was written in C for both applications. It was also noted that the bar code scanner took 30ms to scan a bar code. These variables are common to both applications. The two applications used a different microprocessor, which are supposed to be the same speed, however this has not been proven. This was never checked because it was never considered until months after the test was performed. No retest considering this was performed, as for other reasons the whole test was considered invalid. A possible reason for the large discrepancy was that the program execution

flow differs between the applications. The C based application was procedure based with one thread running the whole application, whereas the Java application was object orientated with multiple threads. To make the test fairer both applications should have used the same methodology where both test were either single or multithreaded. With the resources available this was not an option. For these reasons it was decided that the test did not show a fair comparison between the two languages.

## Bench Mark Testing

With the inconsistency of results from the previous test, a new test was designed. This was designed to test some of the basic operations of a microprocessor, such as; binary operations, mathematics, comparisons and loops. The source code for both the C and Java application was written in parallel so the program execution would be similar. Both applications used the same hardware and microprocessor.

### Test Set Up

Both the C and Java applications used the same hardware platform, the M16 CPU emulator. For the Java tests the Java virtual machine was configured without the debugger interface. The test software, which is listed in appendix K, was designed to test the following operations as listed in the result table 4.3. The timing measurements were taken using an oscilloscope measuring the time delay of an output pin. An input pin was used to trigger the next test.

### Results

The C application was tested twice to get the timing measurement. As the last test application showed that, the time delay with the Java application was inconsistent, five tests were performed with Java and the results were averaged. The C application was only tested twice as there were no interrupt overheads and the execution of the machine code was exactly the same each time. The execution sequence of the Java application was not exactly known, due to timer and garbage collector running asynchronously with interrupts occurring, which were required by the Java virtual machine. As can be seen

with the C tests there were no differences in time delays. However the Java tests do show slight differences.

**Table 4.3 Benchmark test results**

| Test | C application (ms) | | Java application (ms) | | | | | | Speed ratio |
|---|---|---|---|---|---|---|---|---|---|
| Test number | 1 | 2 | 1 | 2 | 3 | 4 | 5 | Average | |
| Binary OR | 4.5 | 4.5 | 500 | 500 | 500 | 500 | 500 | 500 | 111 |
| Binary AND | 4.6 | 4.6 | 500 | 500 | 500 | 500 | 500 | 500 | 109 |
| Binary NOT | 3.8 | 3.8 | 430 | 430 | 428 | 430 | 430 | 430 | 113 |
| Addition | 4.6 | 4.6 | 500 | 500 | 500 | 500 | 500 | 500 | 109 |
| Subtraction | 4.5 | 4.5 | 500 | 500 | 500 | 500 | 500 | 500 | 111 |
| Multiplication | 8.7 | 8.7 | 510 | 510 | 500 | 510 | 510 | 508 | 58 |
| Division | 11.8 | 11.8 | 530 | 530 | 521 | 530 | 530 | 528 | 45 |
| Remainder | 10.9 | 10.9 | 520 | 520 | 521 | 520 | 520 | 520 | 48 |
| Binary left shift | 5.1 | 5.1 | 500 | 500 | 500 | 500 | 500 | 500 | 98 |
| Binary right shift | 5.1 | 5.1 | 500 | 500 | 500 | 500 | 500 | 500 | 98 |
| Compare greater | 3.1 | 3.1 | 400 | 400 | 400 | 400 | 400 | 400 | 129 |
| Compare less | 3.3 | 3.3 | 400 | 400 | 400 | 400 | 400 | 400 | 121 |
| Compare greater & equal | 3.2 | 3.2 | 400 | 400 | 400 | 400 | 400 | 400 | 125 |
| Compare less & equal | 3.2 | 3.2 | 400 | 400 | 400 | 400 | 400 | 400 | 125 |
| Compare equal | 3.2 | 3.2 | 400 | 400 | 400 | 400 | 400 | 400 | 125 |
| Compare not equal | 3.9 | 3.9 | 480 | 480 | 484 | 480 | 480 | 481 | 123 |
| Compare OR | 3.8 | 3.8 | 510 | 500 | 510 | 510 | 510 | 508 | 134 |
| Compare AND | 3.5 | 3.5 | 440 | 440 | 440 | 440 | 440 | 440 | 126 |
| For loop | 10.1 | 10.1 | 1100 | 1100 | 1100 | 1100 | 1100 | 1100 | 109 |

As shown in the result table, excluding the multiplication, division and remainder tests, the application is between 98 and 134 times slower than the C application. The multiplication, division and remainder tests were 45 to 58 times slower. The author could not ascertain a reason for this and decided the overall test results where invalid for reasons discussed below. A speed reduction of 100 meant that an equivalent C program on a 16Mhz processor would need a 1.6Ghz processor for Java programs to perform as well. Further inspection revealed that each test was performed 1000 times while the time measurement was taken. This meant that the loop function (a 'for' loop) was common to all tests. The reason for looping the tests was to reduce the impact of time delays in the I/O routines. The I/O routine time delays were one of the potential

problems that made it difficult to determine the reliability of the SICK application as an evaluation test. The test application was redesigned so each test was only performed once, with the time delay of the I/O subroutines calculated. The time delay was calculated by toggling an output pin of the microprocessor successively. The results with the I/O subroutine calculated and the repeat loops removed, are as follows.

**Table 4.3 Benchmark without the repeat loop results**

| Test | C application (ms) | | Java application (ms) | | | | | | | Speed Ratio |
|---|---|---|---|---|---|---|---|---|---|---|
| | C | C - I/O time | 1 | 2 | 3 | 4 | 5 | Average | Java average – I/O time | |
| IO access time | 0.029 | 0 | 0.9 | 0.9 | 0.9 | 0.9 | 0.9 | 0.9 | 0 | 31 |
| Binary OR | 0.062 | 0.033 | 1.5 | 1.5 | 1.5 | 1.51 | 1.59 | 1.52 | 0.62 | 19 |
| Binary AND | 0.064 | 0.035 | 1.36 | 1.36 | 1.37 | 1.36 | 1.36 | 1.36 | 0.462 | 13 |
| Binary NOT | 0.061 | 0.032 | 1.38 | 1.29 | 1.29 | 1.29 | 1.29 | 1.31 | 0.408 | 13 |
| Addition | 0.064 | 0.035 | 1.36 | 1.36 | 1.36 | 1.36 | 1.36 | 1.36 | 0.46 | 13 |
| Subtraction | 0.064 | 0.035 | 1.47 | 1.36 | 1.45 | 1.36 | 1.36 | 1.4 | 0.5 | 14 |
| Multiplication | 0.073 | 0.044 | 1.48 | 1.46 | 1.37 | 1.37 | 1.48 | 1.43 | 0.532 | 12 |
| Division | 0.073 | 0.044 | 1.39 | 1.5 | 1.39 | 1.39 | 1.41 | 1.42 | 0.516 | 12 |
| Remainder | 0.073 | 0.044 | 1.41 | 1.47 | 1.39 | 1.39 | 1.38 | 1.41 | 0.508 | 12 |
| Binary left shift | 0.066 | 0.037 | 1.36 | 1.35 | 1.45 | 1.36 | 1.36 | 1.38 | 0.476 | 13 |
| Binary right shift | 0.066 | 0.037 | 1.36 | 1.38 | 1.36 | 1.36 | 1.36 | 1.36 | 0.464 | 13 |
| Compare greater | 0.055 | 0.026 | 1.16 | 1.14 | 1.13 | 1.14 | 1.13 | 1.14 | 0.24 | 9 |
| Compare less | 0.057 | 0.028 | 1.13 | 1.14 | 1.22 | 1.14 | 1.13 | 1.15 | 0.252 | 9 |
| Compare greater & equal | 0.057 | 0.028 | 1.13 | 1.14 | 1.25 | 1.14 | 1.13 | 1.16 | 0.258 | 9 |
| Compare less & equal | 0.057 | 0.028 | 1.16 | 1.14 | 1.14 | 1.23 | 1.13 | 1.16 | 0.26 | 9 |
| Compare equal | 0.054 | 0.025 | 1.13 | 1.14 | 1.14 | 1.23 | 1.25 | 1.18 | 0.278 | 11 |
| Compare not equal | 0.061 | 0.032 | 1.37 | 1.35 | 1.34 | 1.35 | 1.34 | 1.35 | 0.45 | 14 |
| Compare OR | 0.061 | 0.032 | 1.37 | 1.37 | 1.36 | 1.37 | 1.38 | 1.37 | 0.47 | 15 |
| Compare AND | 0.061 | 0.032 | 1.3 | 1.3 | 1.3 | 1.3 | 1.3 | 1.3 | 0.4 | 13 |

The results showed that the speed reduction was a factor between 9 and 19. With the exception of the I/O access time, which was 31 times slower. The reason for this was the Java application required access to the native code interface. The test results fitted in with an article the author read stating, "It's been reported that Java is 10 to 20 times slower than C, on average." (Wickham, n.d., p. 1). A reason for the speed reduction ranging from 9 to 19 is that some byte codes take longer to execute than others. There is no clear reason for the 'for' loop being approximately 100 times slower however after

reverse compiling and disassembling the class file, there is an indication that the counter variable used in the for loop was reallocated every loop. Thus garbage collection is possible at the end of each loop.

## Conclusions

Two types of test were performed; one an application, the other a benchmark test. Both tests showed different performance characteristics of Java. The first test was an application, which showed results that were not consistent with other research findings. The second test application showed results consistent with other research findings. The first application showed that in one instance the Java application was potentially only 2.5 times slower. This however did not take I/O access routine times into consideration. The second test application showed results consistent with other research, however it was not a practical application.

Researching other options for how to develop a better test application a journal article evaluating performance between C and Java stated the following. "Comparing a highly object based Java implementation with a non-object oriented C or Fortran would be hard to interpret, as we cannot readily distinguish between differences due to language implementations and differences due to abstraction penalties." (Bull et al., 2003, p. 418). A further quote, "Of more interest are a number of benchmarks which focus on determining the performance basic operations such as arithmetic, method calls, object creation and variable accesses. These are useful for highlighting differences between Java environments, but give little useful information about the likely performance of large application codes." (Bull et al., 2003, p. 419). The first quote discusses that it is hard to develop a test application comparing an object orientated program with a procedure based program, which is essentially what was tried in the first test application. The second quote discusses that testing basic arithmetic functions as performed in the second test application does not show the performance characteristics of a large application. Bull et al. further discusses and evaluates the use of the Java Grande benchmark applications. Grande applications are discussed as; "A Grande application is

an application, which has large requirements for any or all of: memory, bandwidth and processing power. Grande applications include computational science and engineering codes, as well as large-scale database applications and business and financial models." (EPCC, 2005). Looking further into Grande benchmarking, it is based for scientific applications, calculating Fourier coefficients, and matrix-vector multiplication. This is not necessarily valid for automation and control, however highlights some of the problems with the test applications.

The above findings from Bull et al. indicate that the first test application is the better way of evaluating performance. To further improve the test application the following should be considered.

- The effects of the garbage collector.
- Timing measurements taken on how long I/O subroutines/methods for native C code that is common to both the C and Java applications.
- Every aspect of the application should be tested, in regards to timing.

The garbage collection is relatively random and is most likely the cause for the large spread of results across the tests in the first application. This is a large drawback with using Java for real-time automation and control, however there is the possibility of reducing it effects by careful software programming such as; Avoiding the creation and deleting of objects while the application is running, and calling the garbage collector manually while the program is waiting or performing non time critical tasks.

Timing measurements need to be taken on the access times of native I/O subroutines that are common between both tests. This is required so that only the application code is compared removing elements of software and hardware that are common to both applications. In the first application a large communication engine written in C was common between both the Java and C applications. The time spent processing the

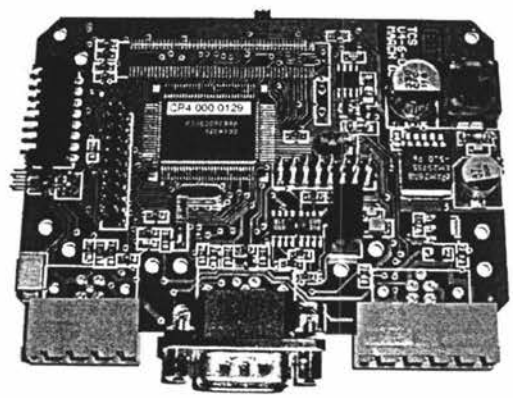
communications data was not removed from the timing measurements; this could have resulted in the large differences speed between application and tests.

The application should consist of multiple tasks, with each task being tested and the results between each task analysed to find out an average speed difference throughout the whole application. This would approximate an overall view on the performance across the two applications, reducing the effects of the different program structures between object orientated Java and procedure based C.

# Chapter 5 Hardware Platform Development for Function Block Programming

Initially the developer expected to use their existing CPU4 platform (CPU4 is their designation for the hardware platform in their current product range). The CPU4 was developed a year before the commencement of this project, to supersede an 8 bit CPU3 platform. During development of the CPU4 platform the developer decided to use a 16 bit Mitsubishi microprocessor, with the expectation of the platform being suitable for IEC 61499 function block software. As discussed in previous chapters, the requirements of Java meant that the CPU4 platform was not suitable. In the early stages of the project, work commenced on porting the KVM to the CPU4. This was done to find out whether Java was suitable for use with embedded platforms. From the findings discussed in previous chapters, this has been verified. However, the increased processor power required meant the CPU4 was inadequate as a production system.

**Figure 5.1 CPU4 target platform before memory expansion modifications**



The dimensions of the CPU4 motherboard are 100x70mm.

## Choice of Target Microprocessor

While porting the Java virtual machine to run on the CPU4 platform, it was noticed that the use of processor resources was increasing. Bernard Wood, a colleague, (personal communication, 2003) performed some research for a suitable 32-bit microprocessor. He looked at different microprocessor options, such as the Mitsubishi M32, an upgraded version of M16, and the ARM microprocessor which is typically used in PDA's.

In the interests of building commercially viable products for the developer, Wood looked at microprocessors that were commonly used in consumer products. He investigated support, availability and cost of components. While he was performing his research he was unable to determine a suitable specification for the microprocessor. He could not find anyone who had used Java in the way that the developer planned. The only person the author and developer could find using an embedded microprocessor for function block programming with Java, was Dr Jim Christensen (personal communication, September-October, 2003) from Rockwell Automation. Christensen's platform was the Imsys SNAP. This platform comes complete with a Java virtual machine built into it. This platform was not chosen for this project because it was not a suitable hardware format for a final product and there was no support for the use of the developer's DeviceNet communications software that was written in C. The developer's DeviceNet communications software is an important part of their intellectual property. There did not appear to be any investigation into why the SNAP was used, other than that it was a pin for pin replacement for Christensen's previous development platform the TINI. Christensen had said that the TINI was not powerful enough to run IEC 61499 function blocks properly (J. H. Christensen, personal communication, September-October, 2003).

It was concluded that the only practical way of finding out what was required in terms of processor resources, was to get a function block run-time environment running and evaluate its performance. This meant that function block development had to be

performed on the developer's current CPU4 platform, however there was a shortage of memory space with the M16 microprocessor. A M32 version of the processor which is a pin for pin replacement for the M16 and has more memory space, was considered. The developer could not pursue this option due to the merger of Mitsubishi and Hitachi resulting in costs of development equipment increasing beyond the developer's budget, and component delivery lead times exceeding the time frame required for this research project.

It was concluded that at this stage in the project, that the exact requirements for the microprocessor were unknown. The developer was reluctant to spend large amounts of money developing a new platform while not knowing its full specification. Because of this, a decision was made to expand the existing CPU4 platform for a proof of concept application for IEC 61499 function blocks.

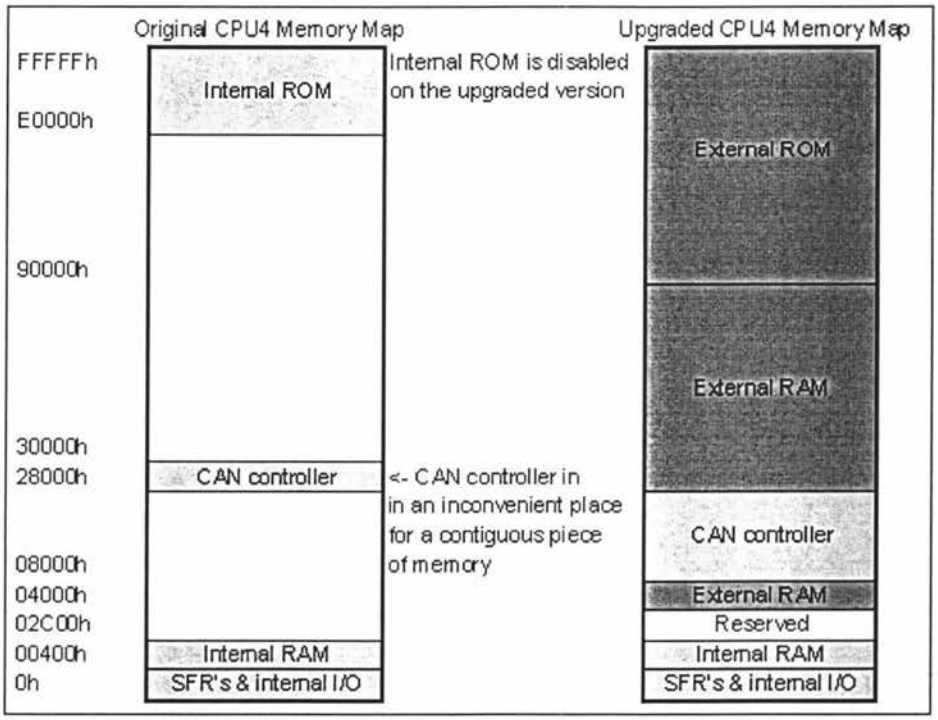
## Increasing the Memory of the CPU4 platform

Initially the Java virtual machine and the IEC 61499 run-time environment were tested using a M16 ICE (in circuit emulator) which this emulates a M16 microprocessor and is attached to the CPU4 motherboard. The M16 emulator has 1Mbyte of RAM, of which some is used for ROM. The target platform using the actual M16s has 120kbytes of ROM and 10kbytes of RAM. The Java virtual machine uses 448kbytes of ROM, the Java application uses approximately 500kbytes of ROM and was requiring 160kbytes of RAM just for testing interface function blocks. This itself was requiring more than the 1Mbyte of address space that was available on the M16. Initially the Java application was reduced to get the code size to fit into the M16 ICE, however this was not practical when a full application was required.

It was noted that when the Java virtual machine reads data from the class files, it does not directly address memory. Instead it uses an external subroutine to get the class file data. This meant the Java application could be saved to some external storage device without too much difficulty. A flash chip with an SPI interface was chosen. SPI is a

serial communications interface standard. A 2Mbyte chip was used, providing the required space for the large Java application. The addition of the serial flash allowed more effective use of the 1Mbyte address space, allowing the whole Java application to be downloaded to the target platform. However, the target platform still required a further memory upgrade unless the 'In circuit CPU emulator' was used. This required development of a memory expansion board. One of the staff members of the developer company started the development of the serial flash and memory expansion board, but before the hardware was built he left the company. This departure resulted in the author completing hardware development. Before the requirements of the memory expansion hardware could be defined, a memory map was planned. One of the problems was that the Java virtual machine required a large contiguous section of memory. The CPU4 platform had the CAN chip (the DeviceNet hardware controller chip) mapped into the device in an inconvenient section of the memory map.

**Figure 5.2 CPU4 Memory Maps**




As shown in the above memory maps the CAN controller was mapped into the middle of a large empty section of the CPU4's original memory space. This reduced a large

contiguous section of memory that was required by the Java virtual machine. A decision was made to shift the address of the CAN controller so its start address would be 8000h and only map the 32 bytes required by the CAN controller. To do this, a PLD (Programmable Logic Device) was used for all the address and memory decoding.

The PLD performed well for the memory address decoding and switching the chip enable lines of the memory chips. However the PLD program was not written until after the first memory expansion board was made. This resulted in PLD limitations not being considered. The biggest limitation was the PLD bi-directional I/O ports. The original hardware design had the data bus going through the PLD, this is where the data bus is not directly connected to the memory chips. To achieve this the data bus path in the PLD must be likened to a wire, where data can flow both directions. The PLD bi-directional ports can only send data one direction and require switch circuitry to switch them to the other direction. With this problem it was decided to reduce the amount of external memory. From this, it was concluded that a PLD could simplify hardware design, however the PLD program should be written in conjunction with the hardware circuit diagram.

As there were a number of errors and the with hardware designer's departure from the company, the hardware fault finding and correction was done by the author.

## Products that Required Converting to IEC 61499

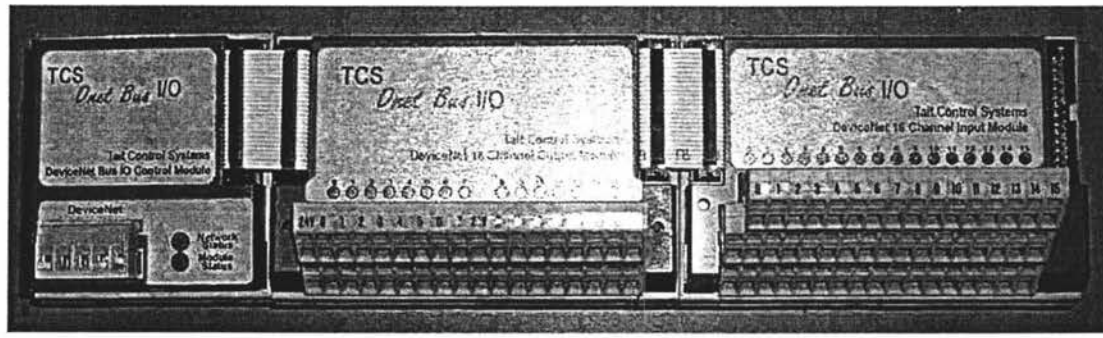
The final task of this research was to build a distributed control application, that was evaluated for this research and demonstrated at the EMEX show, Auckland, May 2004. The demonstration at the EMEX show was to create market interest in the developer's latest product developments and technology. The developer suggested the use of the following products for the show: a serial communication interface, a 16 way input/output module, and a motor control station. The serial communication interface required no further hardware development as the CPU4 motherboard has an on board serial port. However, the 16 way I/O module and motor control stations require

additional hardware, as the CPU4 motherboard does not have the I/O circuitry required. Modification and development was required for some of these circuits because they were incompatible with the CPU4 motherboard. Further details of the demonstration machine are discussed chapter 6.

## Upgrading the 16 Way I/O Module for Distributed Control Technology

For the show machine, discussed in chapter 6, a 16 way I/O module was required. As the developer's intention was that only one I/O module be built, for demonstration purposes, they chose to have an obsolete bus I/O product modified to suit the new technology. The bus I/O module is shown in the figure below, it consists of a control module, 16 output module and a 16 input module. The I/O module was later used to control the conveyor and three axis robot discussed in later chapters.

**Figure 5.3 Original bus I/O product**




This used parallel bus architecture to communicate with input and output modules with an 8051 microprocessor. The bus architecture of the 8051 and the M16 microprocessor used in the CPU4 were not compatible. To avoid complete redesign of a product, the software of the old product was modified to use serial communications instead of DeviceNet. The serial port was then interfaced to a CPU4 motherboard and packaged together as one product. This interface was effective and solved the slow scan cycle problem of reading the digital inputs. The serial communications hardware contains a buffer that buffered all the short I/O pulses that the CPU4 inputs and output hardware would miss.

## Upgrading the Motor Control Station

The motor control station contains a three-phase contactor (relay), circuit breaker, and isolator with a network interface to a PLC. There is also some digital I/O. The PLC would control the three-phase contactor, the digital outputs and receive the status information. The upgrade consisted of shifting the control functionality from the PLC to the motor control station using function blocks.

The memory upgrade hardware on the CPU4 platform uses the interface ports that were required by the motor control station hardware. To solve this problem a PLD was used to act as an I/O expansion device. The motor control data was sent between the microprocessor and PLD in serial format. The PLD would then send the data in parallel format to the motor control station hardware. The PLD successfully provided the required functionality.

# Chapter 6 Selection and Development of the IEC 61499 Run-time Environment

## Selection of an IEC 61499 Run-time Environment

The IEC 61499 run-time environment is the term used for the interpreter that reads and executes the IEC 61499 function block diagrams. At commencement of this project, the only run-time environment available was the Function Block Run-time environment (FBRT). This is a fully working IEC 61499 function block run-time environment developed by Rockwell Automation that runs on the PC and the Imsys SNAP embedded Java development platform.

This gave the author and developer two options; either purchasing the source code and software licences of the FBRT and modifying it to suit the developer's target platform, or completely developing a run-time environment. The decision of whether to purchase or develop was made in conjunction with the developer, who decided the best commercial option. The author's input was to estimate the time required to develop a run-time equivalent to the FBRT. After investigating the requirements for developing an IEC 61499 run-time environment, an estimation was made of the time it would take to achieve the milestones required.

The stages in developing a run-time environment are listed below:

1) Planning of the internal structure of a run-time environment.
2) Establishing communications with the software development tools.
3) Building the interface connections between function blocks and scheduling of task execution.
4) Development of entry level function blocks, such as math functions, timers and counters, which can build a real world application.
5) Building applications to test the basic function blocks.

6) Writing service interface function blocks for communications with I/O devices.
7) Development of specialist function blocks such as adaptors and composite function blocks.
8) Build a test application which can test all of the above.

From this plan it was estimated that it would be midway through 2005 before a run-time environment would be viable. A report was made from these estimates and presented to the developer, a copy of which is included in appendix D.

Rockwell Automation's run-time environment offered the following:

- Reduced amount of planning for development of service interface function blocks.
- The interfacing and scheduling of function block tasks already developed.
- Entry level basic function blocks that perform mathematical operations, timers and counters, etc.
- Compatible with the FBDK (Function block development kit, the software tool that is used for developing IEC 61499 programs). This makes development of future function blocks simpler.

Modifications required by the developer to the Rockwell Automation run-time environment are listed below:

- Porting the run-time environment source code to use the developer's target embedded platform.
- Modification of the communications interfaces to use DeviceNet instead of ethernet.
- Development of service interface function blocks to communicate with the developer's target platform I/O.
- The addition of basic function blocks to suit the developer's target market.

From the above information and with an acceptable price the developer made the decision to purchase Rockwell Automation's run-time environment (FBRT).

## Function Block Run-time Environment (FBRT)

As part of the agreement between the developer and Rockwell Automation for their IEC 61499 function block run-time environment, a week's training from Dr Christensen was included. Christensen is the main researcher in the development of the IEC 61499 standard at Rockwell Automation, the developer of FBRT, and FBDK. The FBRT was originally written for the PC and then modified to run on an Imsys SNAP target Java platform. The Imsys SNAP is an embedded microcomputer that runs a Java virtual machine and is suitable for prototype product development. Dr Christensen's training covered:

- How to use the FBDK to develop function block programs that run on the PC, and how to program function block programs.
- The basics of the internal structure of the FBRT, such as how the function block algorithms and interfaces work, and how the function blocks communicate between each other.
- How to make service interface function blocks for communicating with I/O devices.
- What steps the developer and the researcher should take to get a product commercially acceptable.

The information gained from this training visit up-skilled the author and the developer with information to give assistance with FBRT development. The tasks that needed completing are listed below:

- Porting the FBRT to run on the developer's target platform.
- Running a simple application to test the basic FBRT.
- Developing communications with the FBDK software.
- Developing communications for Master/Slave communications with DeviceNet.
- Modify the FBRT so it is commercially acceptable.
- Develop a communication interface for peer-to-peer over DeviceNet and build a distributed control application.

- Modify the communications between the FBDK and FBRT to use DeviceNet to allow configuration of distributed control applications.

## Porting the FBRT for use with the Developer's Target Platform

Porting of the FBRT to get a basic application to run on the developer's target platform required only minor modifications, as the FBRT is written in Java. The Imsys SNAP platform uses the Sun J2ME CLDC Java virtual machine which the developer uses, however Imsys has added extra functionality to the J2ME CLDC library. This required disabling the ethernet function block internals. They were later modified for DeviceNet communication.

The 1 Mbyte memory limitation of the developer's target platform now became apparent. Finding a suitable upgrade was difficult. When compiling programs with Java, unlike C, it is hard to find what resources (memory and speed) are required for the application. The ROM usage of Java can be determined by reading the file sizes of the Java application. Determining RAM usage is difficult. The only method of doing this was by querying the virtual machine to find out how much memory it was using. Unfortunately this requires estimating how much memory the application is going to use before it is run. For initial development the application was given 152kbytes. To fit the FBRT into the limited ROM space in the early stages of development, most of the function blocks were removed, meaning it could only run very simple applications.

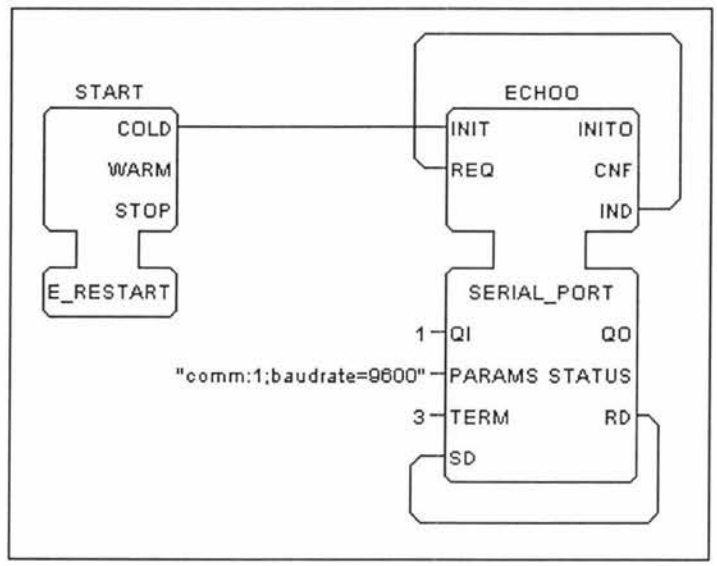
Estimating the speed required was difficult. Christensen said "originally I used the TINI platform before I upgraded to the SNAP. The TINI platform was too slow and the SNAP was quick enough." (J. H. Christensen, personal communication, September-October, 2003). Looking into the specification of the two platforms Christensen had used, the TINI platform using an 8 bit microprocessor that has a speed of 10MIPS, and the SNAP with a 32 bit microprocessor 165MIPS. The developer's platform has a speed of 8MIPS

using a 16 bit microprocessor. MIPS is a term used for processor speed comparisons however is only an indication as it does not take into fact the efficiency of the microprocessor instruction set or data size of the microprocessor. This indicates that developer's platform had a lower MIPS rating than the TINI but is a 16 bit microprocessor, this indicating that it is similar in speed to the TINI when comparing it to the SNAP. When examining the specifications of the TINI and the SNAP it was noted that the SNAP is a drop in replacement for the TINI. This meant that when Christensen upgraded his platform he may not have done research into what speed he needed, other than the fact the SNAP was a lot faster. For initial development the 8MIPS processor was used to gauge the performance requirements.

## First Test Application to get the FBRT Running on the Developer's Target Platform

During Christensen's training visit he built a serial communication function block (described in detail later on) to run on the SNAP platform. From this a simple serial communications loopback program was created and tested on the SNAP. The program is shown in the figure below.

**Figure 6.1 Serial loop back program**



All function block programs start with an E_RESTART function block. This function block acts as the 'Power on reset circuit'. The 'COLD' event refers to a cold boot of the microprocessor. The 'COLD' event output is used to trigger the 'INIT' of the SERIAL_PORT function block. When the SERIAL_PORT function block receives an INIT it uses the data from QI (enable), PARAMS and TERM to configure the serial port hardware. TERM is the terminating character of the input serial data string. SD is the serial data output, (input of the function block) and RD is the serial data input (output of the function block). When serial data is received an IND event is triggered on the output. The IND event is connected to the REQ event. When there is a REQ event then the hardware is told to transmit the serial data that is at the SD input.

This application was first tested using the Imsys SNAP platform because the platform was known to work. The application was then converted into a resource and device type. This is where the application is converted into a Java class file embedding the application into the run-time environment. Converting the application into a resource and device type embeds the application into the device. This means that the FBDK does not need to configure the device every time it boots up. This method of creating function block programs is not suitable for a final product, as it is labour intensive, however it is a simple method to test the run-time environment when there is no communications with the FBDK.

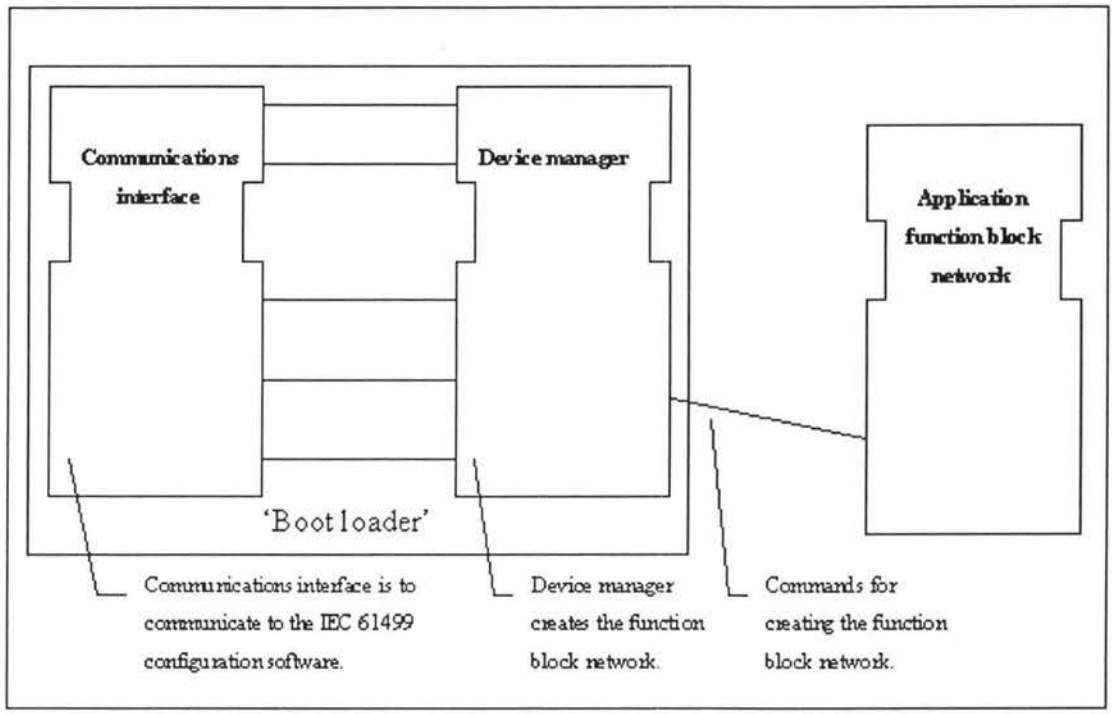
When this application was tested on the SNAP and then on the developer's CPU4 platform, the application worked. When checking the time delay between receipt of data on the serial line and its retransmissions, it was found that the SNAP was twenty times faster than the CPU4.

## Internal Structure of the FBRT

Before explaining the modifications to the IEC 61499 run-time environment, the internal structure needs explanation. The IEC 61499 function block programming language is not implemented as an interpreted language but as a collection of objects

and classes built with common input and outputs interfaces. The common interfaces cause the different objects to communicate and trigger each other. A standard function block device contains a 'Device Manager' that acts as a boot loader, and is responsible for creating the communication links between function blocks. This is explained in the figure below.

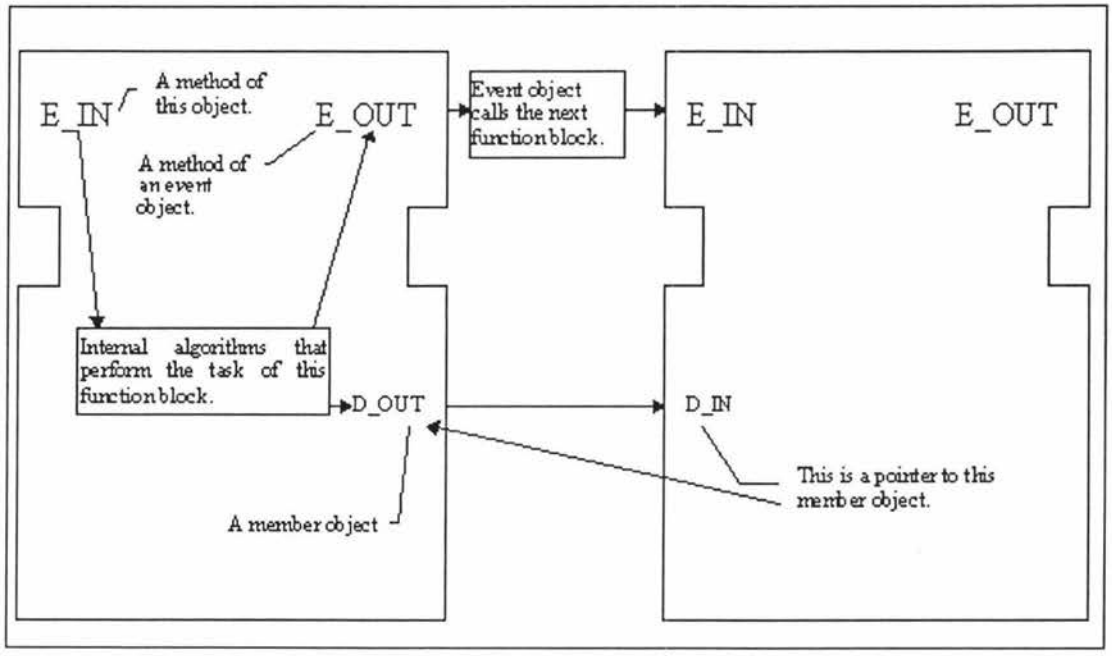
**Figure 6.2 Internal operation of the FBRT**




The boot loader application contains a manager and a communications interface. The communications interface acts as the interface between the configuration tool (FBDK) and the device manager. The device manager receives commands from the communications interface and creates the application. At present applications are volatile, requiring them to be downloaded every time the device is powered up. There is provision to make the applications non-volatile, but this is not in the scope of this research project.

The figure below shows the interconnection structure of function blocks.

**Figure 6.3 Interconnection of function blocks**




The function blocks are all built with standard interfaces between them. There are two types of interfaces: data and event interfaces. A data interface object typically contains a data member and other members such as identifiers. All data outputs are objects, whereas a data input is a pointer to the output object that the input is attached to. An event output object contains a list of the event inputs that it is attached to. When a 'serviceEvent' method of the event object is called, the event object goes through its list of objects (function blocks) that it needs to call and calls the methods inside them. When an event input is called, the algorithms of the function block are executed. With Java's multi-threading capabilities many function blocks can execute simultaneously.

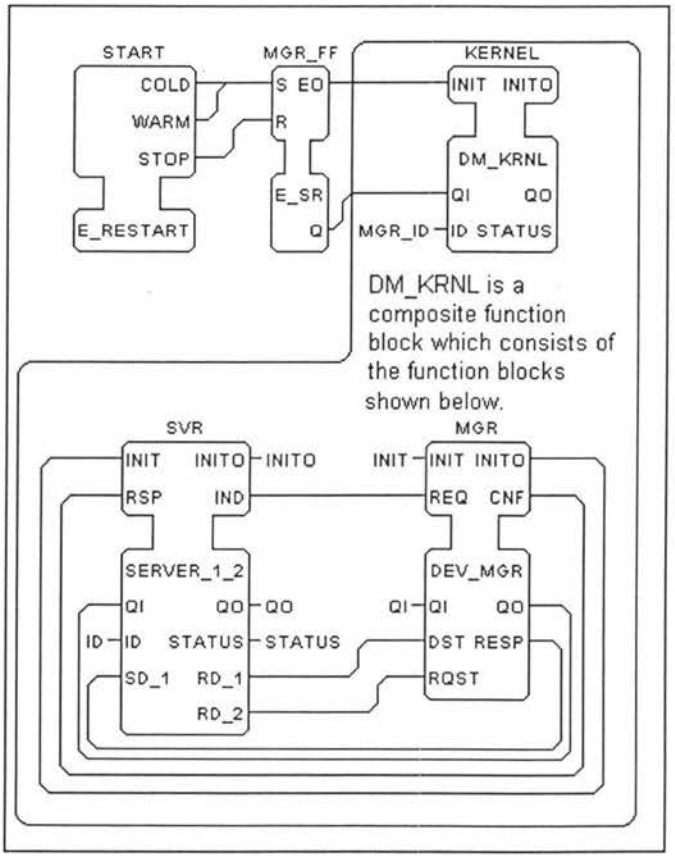
## Developing Communications with the Function Block Development Kit (FBDK)

The Function Block Development Kit (FBDK) is the graphical software tool that is used for drawing function block diagrams. When using the SNAP platform the function block program is downloaded using this software. The communications interface between the FBDK and the SNAP is over ethernet. The developer's CPU4 platform

supports serial and DeviceNet communications, but not ethernet. In later stages of the project DeviceNet is used to program the target platform, however at the beginning of this project upgrading of the developer's DeviceNet communication software was required. To get the run-time package communicating initially, serial communication was used.

For communications with the FBDK the standard device type selected for the target platform is the RMT_DEV. RMT_DEV is the device type used with standard distributed control devices. This contains a 'Manager' resource which could be termed a boot loader sub-application. The internals of this resource are shown in the figure below.

**Figure 6.4 Function block diagram for management resource**

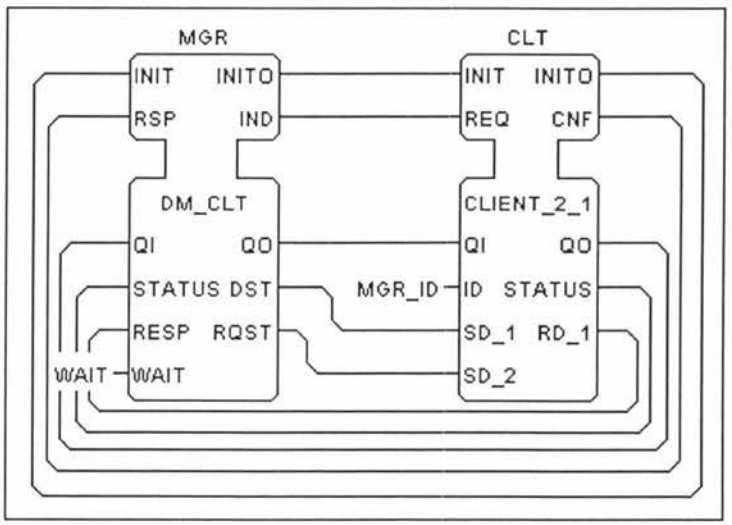



(FBDK, 2003)

Each function block has two names; the name inside the function block is the generic name of the function block (class name). The name above the function block is its specific name (name of the object). The figure shows the E_RESTART triggering a set/reset flip-flop function block communicating with a DM_KRNL function block. The DM_KRNL function block is a composite function block consisting of the function blocks shown in the bottom half of the figure. The DM_KRNL consists of a communication function block (SERVER_1_2) connected to the DEV_MGR function block. The DEV_MGR function block creates the function block programs within the device. The SERVER_1_2 function block is the ethernet communications interface. On the target platform this function block was modified to use RS-232 communications. This task was relatively simple, as Java is object orientated, with only the configuration of the communication interface requiring change.

When a program is downloaded and run using the FBDK, the FBDK on the PC starts a function block run-time environment that acts as a proxy device. The function block program is passed to the proxy device. The proxy device consists of a function block diagram as shown in the figure below.

**Figure 6.5 Proxy device used to down load function block programs into the remote device**




(FBDK, 2003)

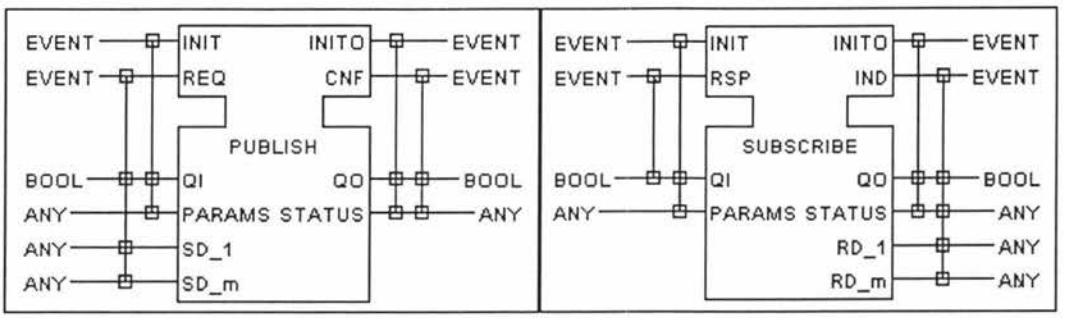
The above figure shows the reverse of the management resource in the target platform. The DM_CLT function block is the interface from the FBDK, and this communicates with the CLIENT_2_1 function block. The CLIENT_2_1 function block is the ethernet interface to the target platform. This function block was also modified for serial communications. This meant that function block programs could now be downloaded to the target platform. This interface worked, however it does not support distributed devices, as RS-232 is a point-to-point protocol.

At a later stage in the project this interface was adapted to use DeviceNet, which is discussed in chapter 7. A serial communication interface to DeviceNet programming adaptor was made as the communications hardware for the PC. This required further modification to the remote proxy. These were performed in a way that the ethernet devices could still be programmed. Using DeviceNet for programming worked, however it was noted that care was required when using a busy network. The reasons for this, is that if the network is being shared with a centralised PLC, the PLC usually has the highest priority on the network, meaning very little bandwidth for the FBRT. This problem can be solved if the PLC is reconfigured with larger network scan delays.

## DeviceNet Communications

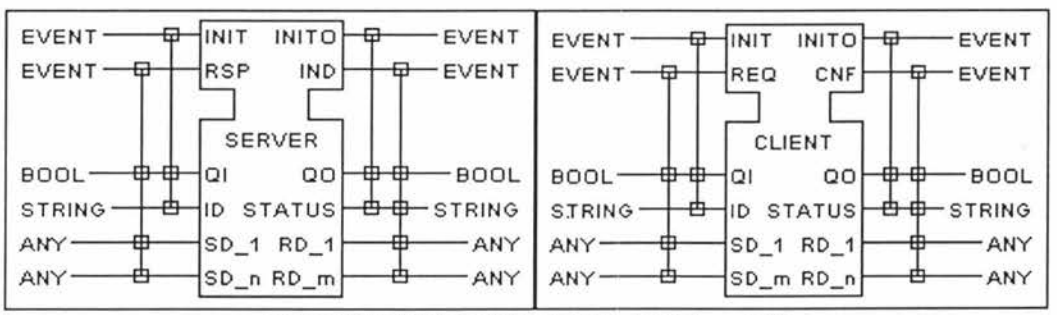
This discusses the Function Block interface for DeviceNet communications. The development of the DeviceNet protocol is discussed in chapter 7. The specifications specify four service interface function blocks for network communications; PUBLISH, SUBSCRIBE, CLIENT and SERVER. PUBLISH and SUBSCRIBE are for multi-cast communications where one node publishes (sends) data and the other node(s) subscribe (receives) to the data. CLIENT and SERVER are used for bi-directional communications, where the same communication link is used for both sending and receiving. The figures below illustrate the function blocks.

**Figure 6.6 PUBLISH and SUBSCRIBE Function Blocks**




(IEC TC65/WG6(PT1CD)4, 2003, p. 97)

**Figure 6.7 SERVER and CLIENT FUNCTION BLOCKS**

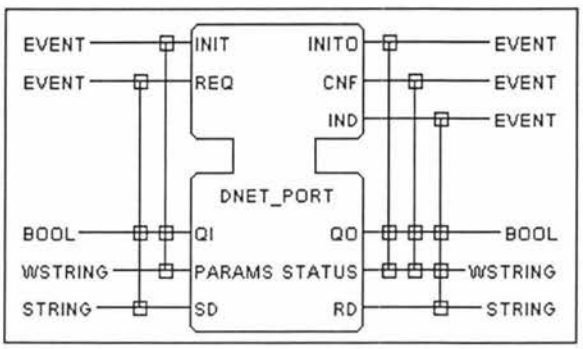



(IEC TC65/WG6(PT1CD)4, 2003, p. 98)

All of the function blocks have the same event and data connections of INIT, QI, PARAMS, INITO, QO and STATUS. The INIT event with QI (enable) and PARAMS (Network connection parameters initialise the network interface) initialise the communication connection. The data that is sent across the communication connection is the SD_m and RD_n data. The REQ, CNF, RSP and IND events are the events that are associated with transmission and reception of data. The data encoding protocol is defined in the IEC 61499 specification, this is a simple protocol of a data type header followed by the data. This protocol is suitable for peer-to-peer communications between nodes as the function block data is encapsulated over DeviceNet. However for master/slave communications with a DeviceNet scanner it is unsuitable, as the packet encoding format is defined by the manufacturer for each product. A reason for the requirement of the master/slave communications is for backward compatibility with commercial systems. It is highly probable that the first applications using function

blocks will be a semi-distributed control system. This will have some of the control distributed, and some centrally controlled by a PLC. This will allow the gradual introduction of the technology into industry to gain user confidence. To overcome this problem specialised function blocks were built for master/slave communications.

**Figure 6.8 DeviceNet Master/Slave Communications Function Block**

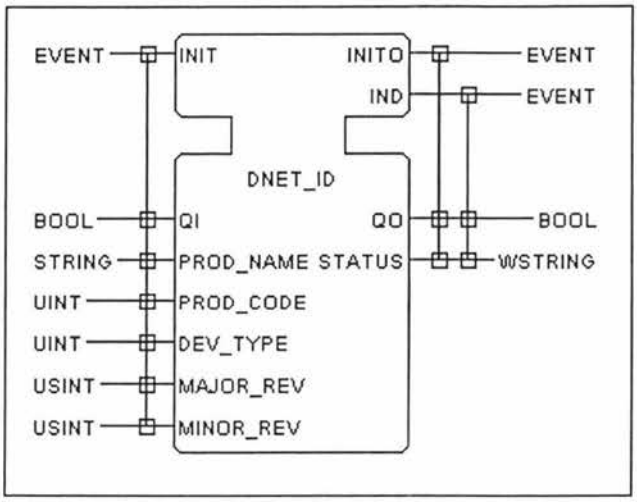



(FBDK, 2003)

Upon initialisation the function block receives an INIT event, QI and PARAMS data. QI is the device enable input, and the PARAMS data is the configuration string. When a REQ event is received the SD data is transmitted on DeviceNet, the CNF event is confirmation that the REQ event has been received. SD data is the input to the function block and the output of the DeviceNet communication software. When data is received from DeviceNet it appears at the RD data output and an IND event is triggered.

With any DeviceNet device it is a requirement that it must have a vendor identifier, device name, product code and revision numbers published in a format defined by the DeviceNet specification. To provide the facility, a service interface function block was created, which is shown in the figure below:

**Figure 6.9 DeviceNet device identification interface**




Upon initialisation the function block receives an INIT event. This copies the device details (PROD_NAME, PROD_CODE, DEV_TYPE, MAJOR_REV and MINOR_ REV) initialising the DeviceNet communications software. On the output side of the function block there is an IND event that indicates the status information of the DeviceNet communication software.

When testing the DeviceNet interface, the peer-to-peer function blocks CLIENT, SERVER, PUBLISH and SUBSCRIBE function correctly, however the DNET_PORT unfortunately requires a number of function blocks to assemble and convert the different data types. The DNET_ID interface for assigning the Vendor information was functional, however a problem arose when the programming interface was upgraded to use DeviceNet. A DeviceNet node must have an Identity object configured before it goes on-line. When using DeviceNet to program a function block device, the DeviceNet network must be on-line. This meant that the DNET_ID function block had to be removed and the vendor information permanently set. A possible way of changing this if required by the developer, is to have a command sequence which can be sent over the DeviceNet medium that allows these values to be changed. This command sequence is considered confidential to the developer The reason for it being considered confidential is the DeviceNet standard setting body, Open DeviceNet Vendors Association (ODVA)

may not allow this feature if it was publicised and is only specific to the developer during product manufacture.

### Master/Slave Communication Parameters

With master/slave devices on DeviceNet, there is a standard interface for device configuration parameters to be configured. To have access to this parameter interface, two service interface function blocks were created. DN_PARAM_IN and DN_ PARAM_OUT, with the input/output direction determined from the network perspective. Two separate function blocks are used for inputs and outputs as some of the input parameters are used for configuration, whereas the output parameters can be used for status information. The parameter function blocks are listed in appendix E. The parameter interface allows the software programmer to add parameters to the device easily. However there maybe an issue with this when the final product is submitted to ODVA for conformance testing, as it unclear in the ODVA specification whether a device has to have all the parameters defined before it goes on-line, as opposed to being dynamically configurable.

## Building the Library of Standard Function Blocks

The FBRT came with the basic function block library defined in the specifications that are listed below:

- Standard event function blocks. "All the standard event function blocks are designed to be used within the definitions of larger user composite function blocks and applications and provide many of the common operations required when modelling event behaviours." (Lewis, 2001, p. 87).
- Information exchange function blocks, for transferring data, between devices and resources.
- Mathematical and logical function blocks.

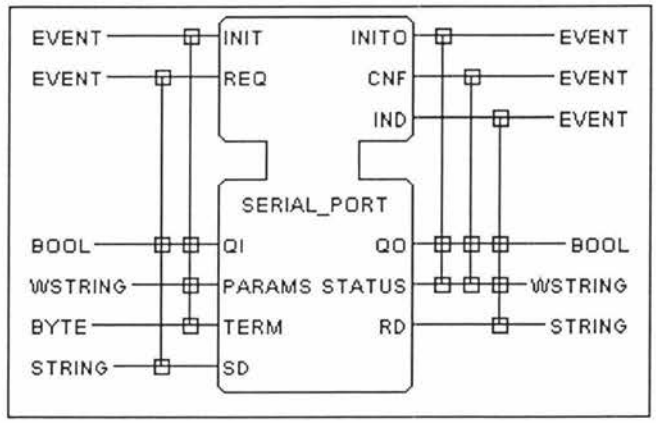
The complete list of function blocks is included in appendix E.

With the development of the run-time environment, consideration has gone into what the most suitable function blocks are for applications. One of the areas in this project that function blocks were created for, was serial communications.

## Serial Communications

The serial communications require specialised function blocks for communicating with hardware and hiding additional protocol overheads such as 'Start' and 'End' characters. The generic serial communications function block was written as a joint effort with Christensen and the author as part of Christensen's training. The function block is shown in the figure below.

**Figure 6.10 Serial port communications interface function block**

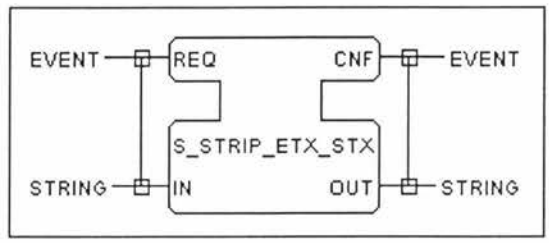



The INIT event reads in the data inputs of QI, PARAMS and TERM that configure the serial port. QI acts as an event request enable bit, PARAMS are the parameters which select and configure the serial port, TERM is the character which terminates the serial port's received data string. To send serial data a REQ event is triggered, and providing the serial port is configured and QI is enabled, the data is transmitted from SD to the serial port. The CNF event is confirmation that the REQ as been serviced. The IND output event is indication that data has been received from the serial port. This happens when data is received by the RD data link. This serial port function block works sufficiently well for the bar code scanner interface application described in a later section. Disadvantages of this interface are that it lacks message time-out facilities if no

termination character is sent. This could result in the interface locking up if the connected device failed to send a termination character. Future improvements could include the reading of set packet lengths and a protocol watch dog timer.

It is common for a serial communication packet to include start and end characters. Here a simple function block was developed which removes the ASCII STX (start) and ETX (end) characters (ASCII code decimal values are 2 and 3).

**Figure 6.11 Function block for removing the ETX and STX characters of a serial packet**




This function block is used for the bar code scanner application, explained in the next section.

## Test Applications

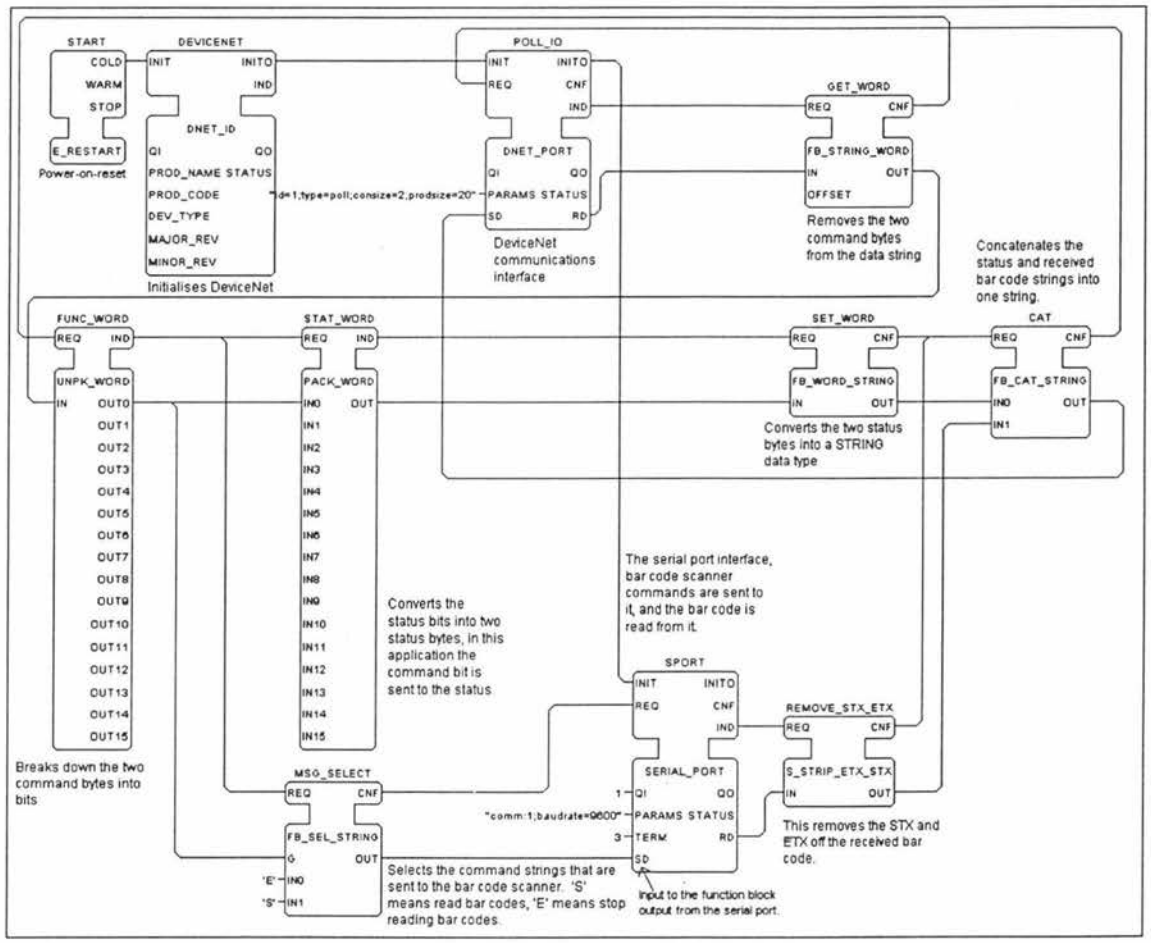
While developing the function block library it was unknown what function blocks needed developing and what a suitable application would be for testing the function block run-time environment. To overcome this problem test applications were developed. The test application showed what was required of the software and hardware evaluating the feasibility of using embedded microprocessors. Two applications were developed: a master/slave centralised device with all its software written using function blocks, and a conveyor and robot system using distributed control.

### DeviceNet Master/Slave Centralised Device Application

The DeviceNet bar code scanner application that was used for the evaluation of embedded Java that was discussed in chapter 4 was chosen, because it could be

compared with the Java and C applications. While developing this application it was observed that to develop the full bar code scanner interface application with all its user configurable parameters was impractical with the developer's CPU4 platform. This was because of the slow processor speed and lack of memory. This meant the bar code scanner interface application was simplified. This application only supported one-shot triggering over DeviceNet and no trigger timers. The bar code size was set to a maximum of 18 characters. The function block diagram for the application is shown in the figure below.

**Figure 6.12 Partial implementation of bar code scanner interface application with function blocks**




The application consists of DeviceNet communication, data processing and bar code scanner communication sections. The POLL_IO function block is the communication

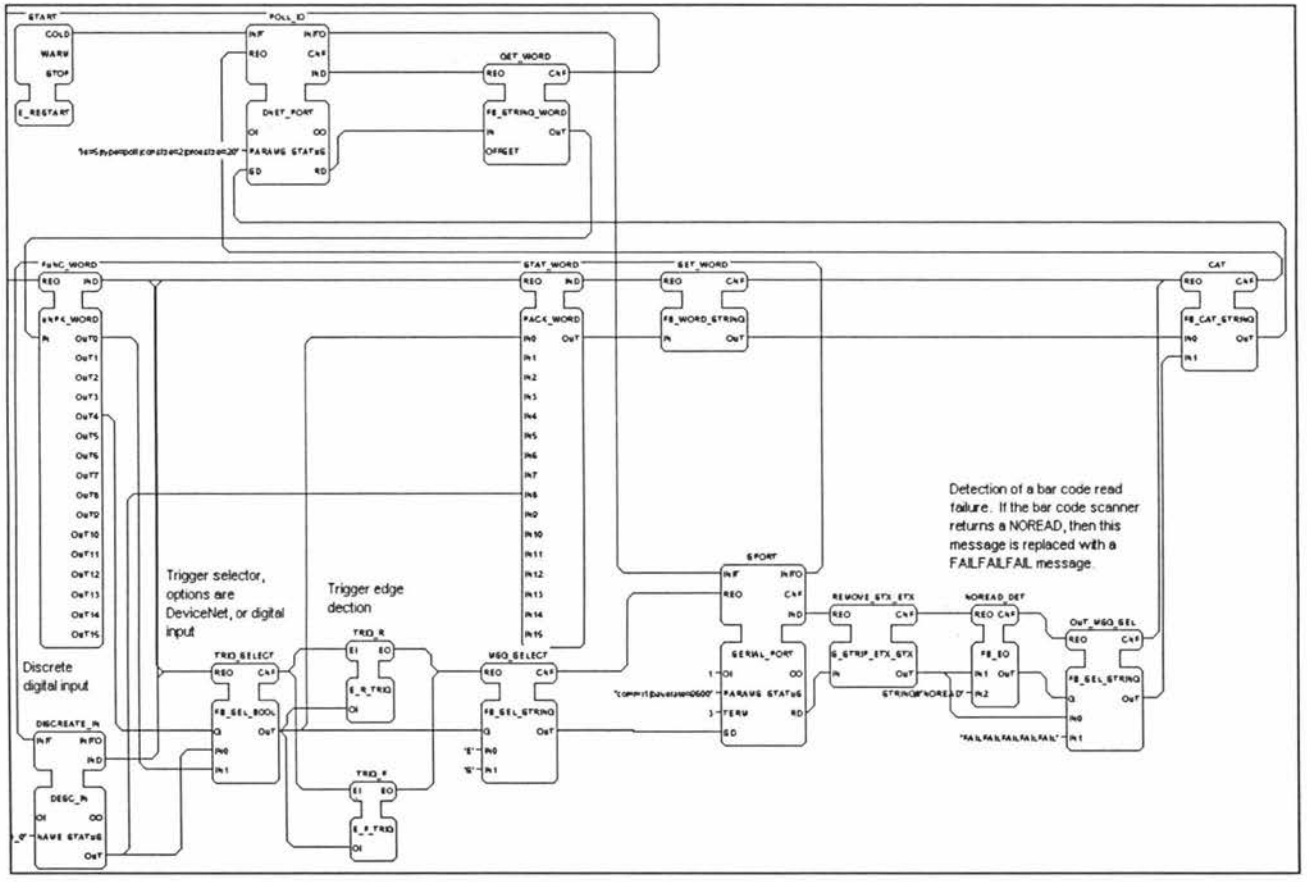
interface to DeviceNet. GET_WORD, FUNC_WORD, STAT_WORD, SET_WORD and CAT assemble and disassemble the DeviceNet data. MSG_SELECT selects the input command to the bar code scanner, an 'S' command triggers the bar code scanner, where an 'E' command stops triggering the bar code scanner. SPORT is the serial port communications interface, the data from MSG_SELECT being sent to SPORT to communicate with the bar code scanner. When SPORT reads a bar code it is sent to REMOVE_STX_ETX which removes the header and trailing characters of the bar code. The event links are set up so the correct devices are connected to each other executing the function blocks in the appropriate order.

Running this cut down version of the bar code scanner interface application proved the concept of writing applications with function blocks and helped to work out the processor requirements. For this application it takes approximately 400ms from the request to read a bar code to the bar code being read. The developer considers 200ms as acceptable. The developer had an option of using a pin for pin replacement microprocessor that had a larger memory address space, however the speed increase was only a factor of three. This would reduce time delay to 133ms. When running a partial application the time delay may be acceptable, but may not be satisfactory for a full application. From this result it gave the developer an indication of what processor requirements are required for the final target platform.

As discussed the current CPU4 platform is inadequate for running a real time control application. To overcome the problem with the limited resources, a partial upgrade on the CPU4 platform was required, along with careful optimisation of the run-time environment. The hardware upgrade is discussed in chapter 5. The optimisation of the run-time environment was performed by examining the code of all processor threads (each processor task) and looking for threads that were polling I/O and potentially wasting processor time. Java does not support I/O interfaces directly, but only via a C code interface that can be called by Java. The C software cannot call a Java software routine using the KVM virtual machine. This limits the use of interrupts, therefore

threads are required to Poll the I/O. In some places polling I/O was the only option as due to hardware limitations interrupts could not be used. In all these threads a 'Thread.yield()' statement was placed. The 'Thread.yield()' statement tells the Java virtual machine to service other threads while waiting. These modifications improved the target platform performance allowing larger function block applications to be programmed. The function block bar code scanner application was upgraded to include external triggering from a digital input. The additional function blocks required are shown in the figure below:

**Figure 6.13 Upgraded bar code scanner interface application**



The additional function blocks for the upgraded application have comments describing their purpose. The upgrade added the extra functionality of selecting between DeviceNet and a digital input for bar code read triggering. Trigger edge detection has also been added to reduce false trigger reads of the bar codes. Additional decoding of the output data from the bar code scanner has been added. This detects a bar code scanner read error and sends a fail message on DeviceNet.

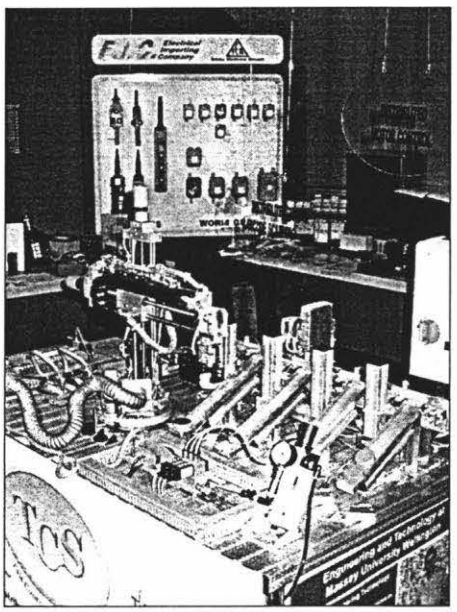
With the addition of the Thread.yields() statements, allowing more efficient use of processor time, the response time improved to approximately 206ms. The 206ms is closer to the developer's specification, however this again was only a partial bar code scanner interface application. Unfortunately even with the increased memory capacity a full application could not be implemented. This indicates that with Java's inefficiency, larger memory and a faster microprocessor is required to achieve the equivalent real time performance with a C based application.

## Conveyor and Robot Application

The conveyor and robot control application was chosen by the developer as a demonstration of IEC 61499 technology at the EMEX show in Auckland in May 2004. The requirement for a show machine also became a feasibility test application for function block programming. The developer company borrowed a FESTO mechatronics conveyor and a three axis robot from the Waikato Institute of Technology. Waikato Institute of Technology use the robot and conveyor for mechatronics and PLC training. The system was modified so the conveyor consisted of a bar code scanner that reads the bar codes on blocks that are passing along the conveyor. From the bar code information, blocks are sorted into four product categories. Photoelectric sensors on the conveyor would sense the product going past and operate the appropriate air solenoids sending the product down reject shoots. The robot would then pick up the products from the reject shoots and place them back on the conveyor. The control system was reconfigured from the use of a centralised PLC to a network of three function block programmed devices. It consisted of a DeviceNet to serial communications interface, a

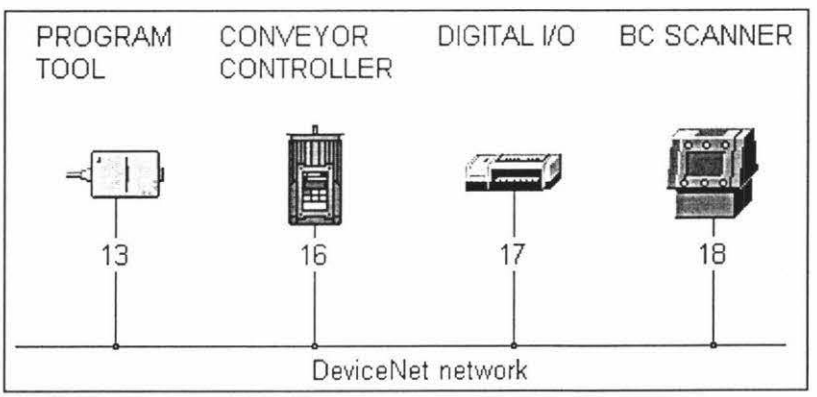
conveyor motor controller and a 16 way I/O module all running IEC 61499 function blocks. The test application is shown in the figure below.

**Figure 6.14 Photograph of test application**



The application consisted of a network of four devices, a programming tool, a barcode scanner (serial communications interface), a digital I/O module and a conveyor controller as shown in the diagram and discussed below.

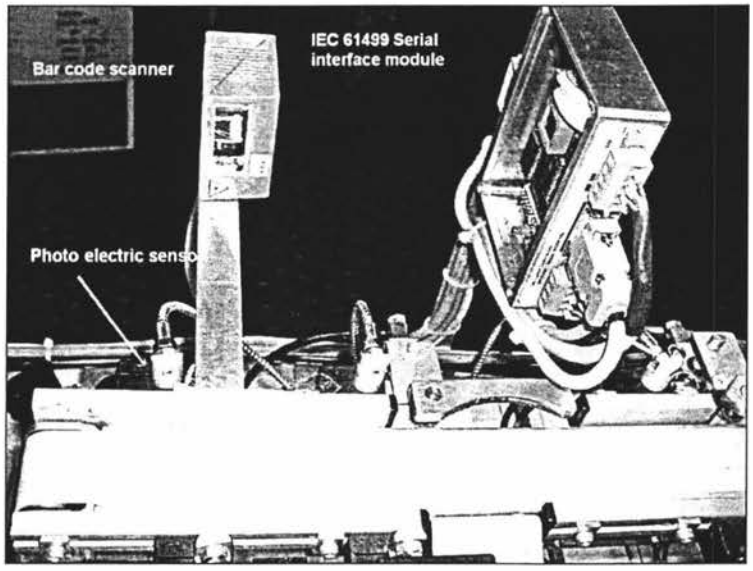
**Figure 6.15 Network diagram**




The programming tool was an interface to DeviceNet using the Function Block Development Kit (FBDK) and was used for configuration purposes only.

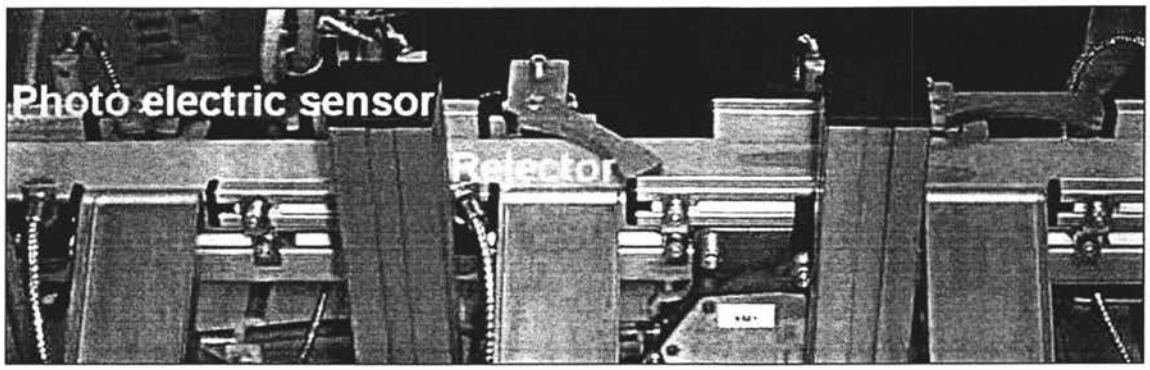
The serial communications interface is the bar code scanner interface. The task of the serial interface was to read a digital input from a photoelectric sensor on the conveyor and send a command to the bar code scanner to read the bar code of the block passing it. When the bar code was read, it would group the blocks into one of four categories; product 1, product 2, product 3, and anything with an unrecognised bar code product 4. The product category information was then passed onto the DeviceNet network using a multicast publisher communications interface.

**Figure 6.16 Photograph of the bar code scanner**



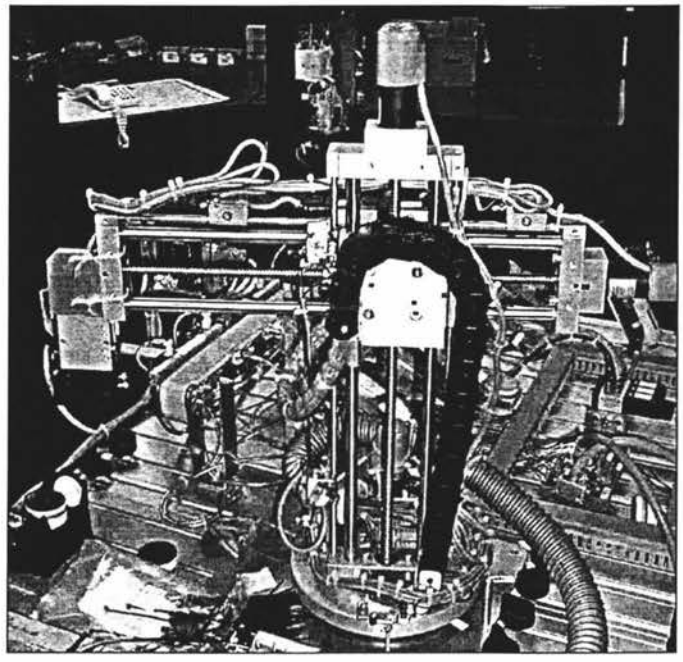

The 16 way I/O module would receive the product category information from the serial interface, and read the status of the photoelectric sensors along the conveyor. When the sensors detected a product passing them, the function block logic in the I/O module would activate the appropriate air solenoid to remove the product from conveyor, this performing the product sorting procedure. When a product is sent down the reject shoot a multicast message would be sent on the network to tell the motor station controlling the conveyor motor to stop. Another message is also sent to the function blocks controlling the robot.

**Figure 6.17 Photograph of conveyor with photo eyes and rejectors**




The robot application was controlled by the 16 way I/O module. The robot would pick blocks up from two of the four shoots and then place them at the start of the conveyor. The robot also has a separate pick up area where a block can be placed for the robot to pick up. The reason for picking up blocks from only two of the shoots is because the robot has limited reach. Once a block is placed on the conveyor, a message would be sent to the motor control station to start the conveyor.

**Figure 6.18 Photograph of three axis robot**



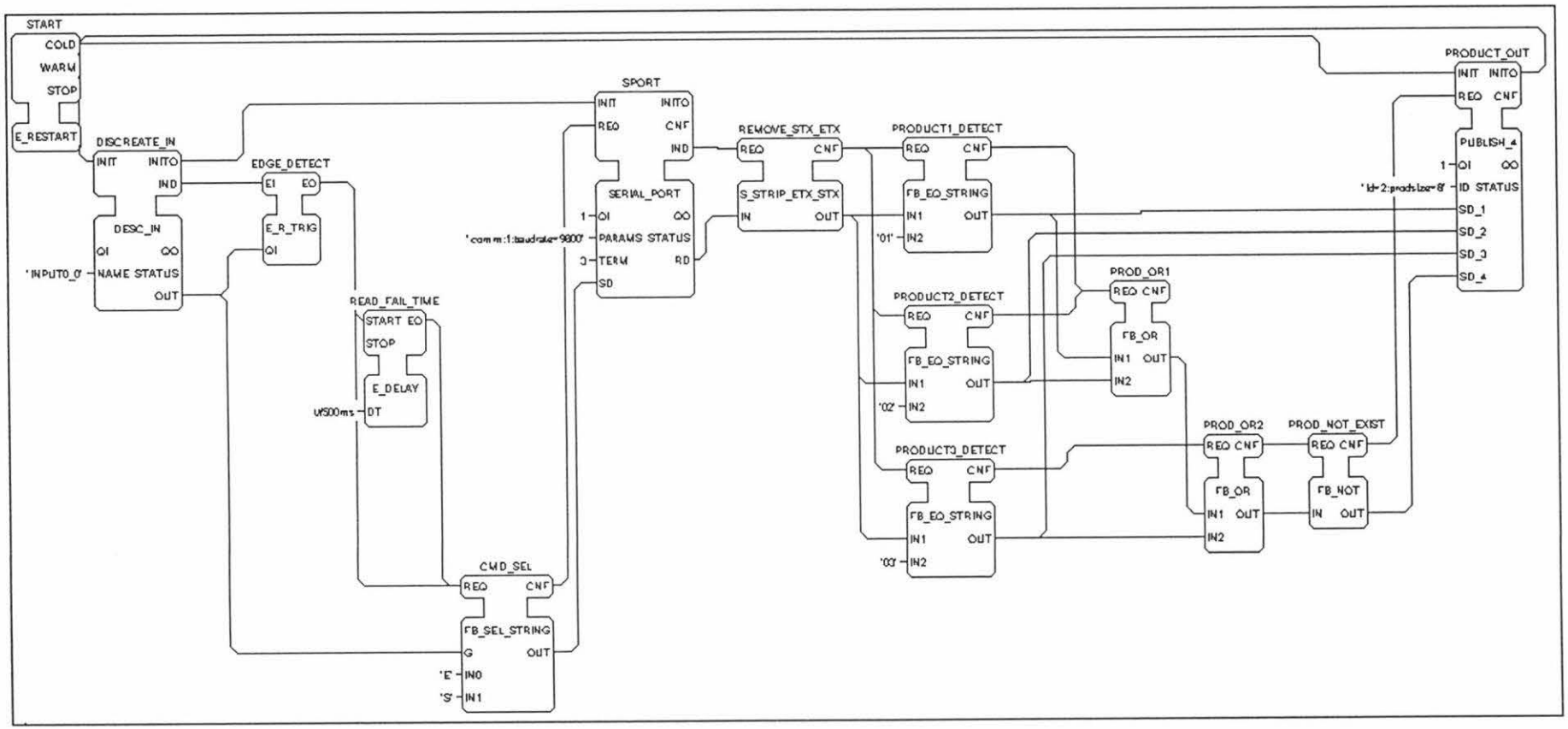
The motor control station consists of a three-phase contactor and circuit breaker with two digital inputs and one digital output. The application software receives information

on whether to start or stop the conveyor, with one of the digital inputs connected to the last photoelectric sensor on the conveyor. The last photoelectric sensor stops the conveyor two seconds after a block has passed it.

## Function Block Software Required

The serial interface was connected to a serial bar code scanner. The software required the use of two specialised I/O function blocks that were not included with the function block run-time environment. These are the serial port interface, and the discrete digital input. The serial interface was the SERIAL_PORT function block discussed previously in this chapter. The discrete digital input was the DESC_IN function block, which included a thread that would poll the digital input, and if the value changed it would send an IND event. The function block diagram is shown below:

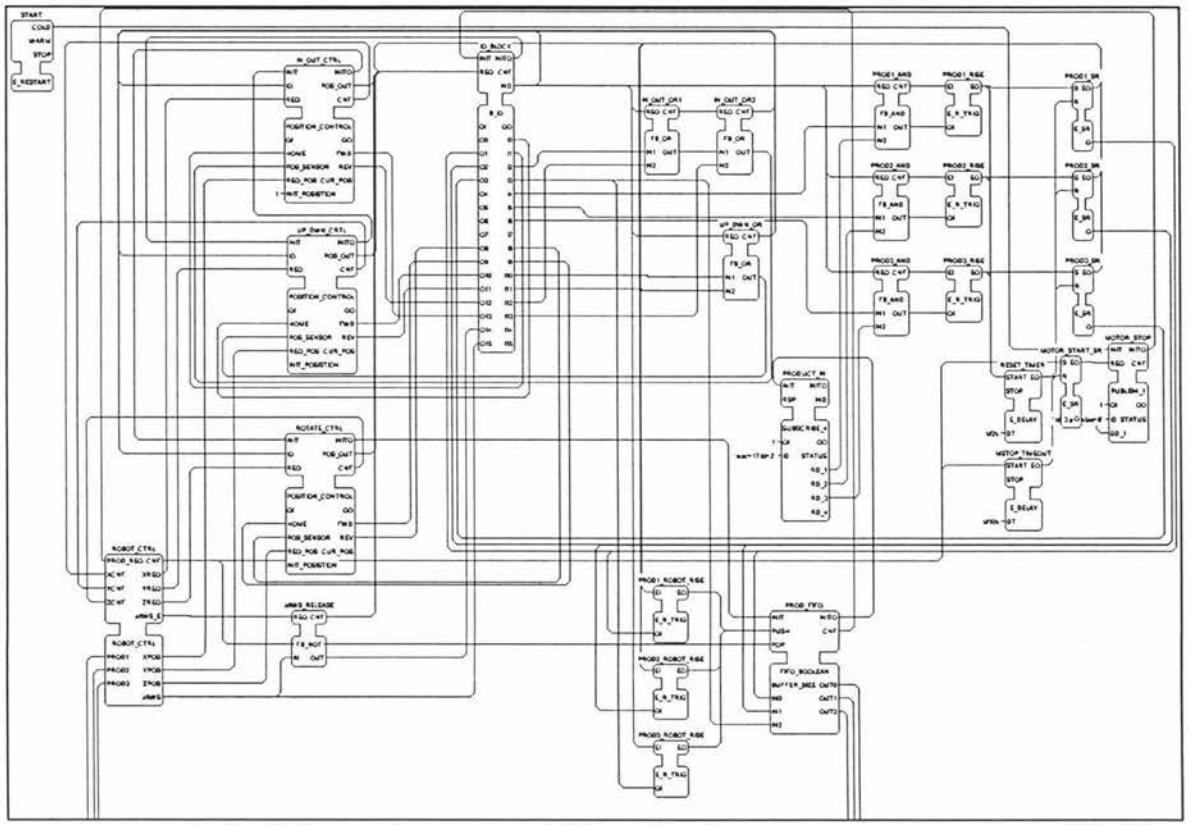
**Figure 6.19 Bar code scanner software**



The input data from DESC_IN is fed into some level triggering function blocks (E_R_TRIG) and a timer function block (E_DELAY) the E_R_TRIG connects to FB_SEL_STRING. This selects the command that is sent to the bar code scanner, 'S' for reading a bar code and 'E' to abort bar code reading. The E_DELAY triggers the event to tell the FB_SEL_STRING to abort reading the bar code. The output of FB_SEL_STRING is fed into the SERIAL_PORT function block. The output from this is fed into S_STRIP_STX_ETX, which removes the start and end characters of the bar code string. The output is fed into three compare function blocks (FB_EQ_STRING), that sorts the products. The bar codes that are being compared are '01', '02' and '03'. The compare function blocks are connected to other function blocks performing some binary logic functions for product sorting. This is then fed into a PUBLISH function block, publishing data on DeviceNet.

The 16 way I/O module required the largest IEC 61499 function block diagram as shown in figure 6.20 as it controls both the conveyor air solenoids and the three axis robot. The large function block near the centre of the diagram is the specialised function block that controls the I/O.

**Figure 6.20 Sixteen way I/O module function block program**

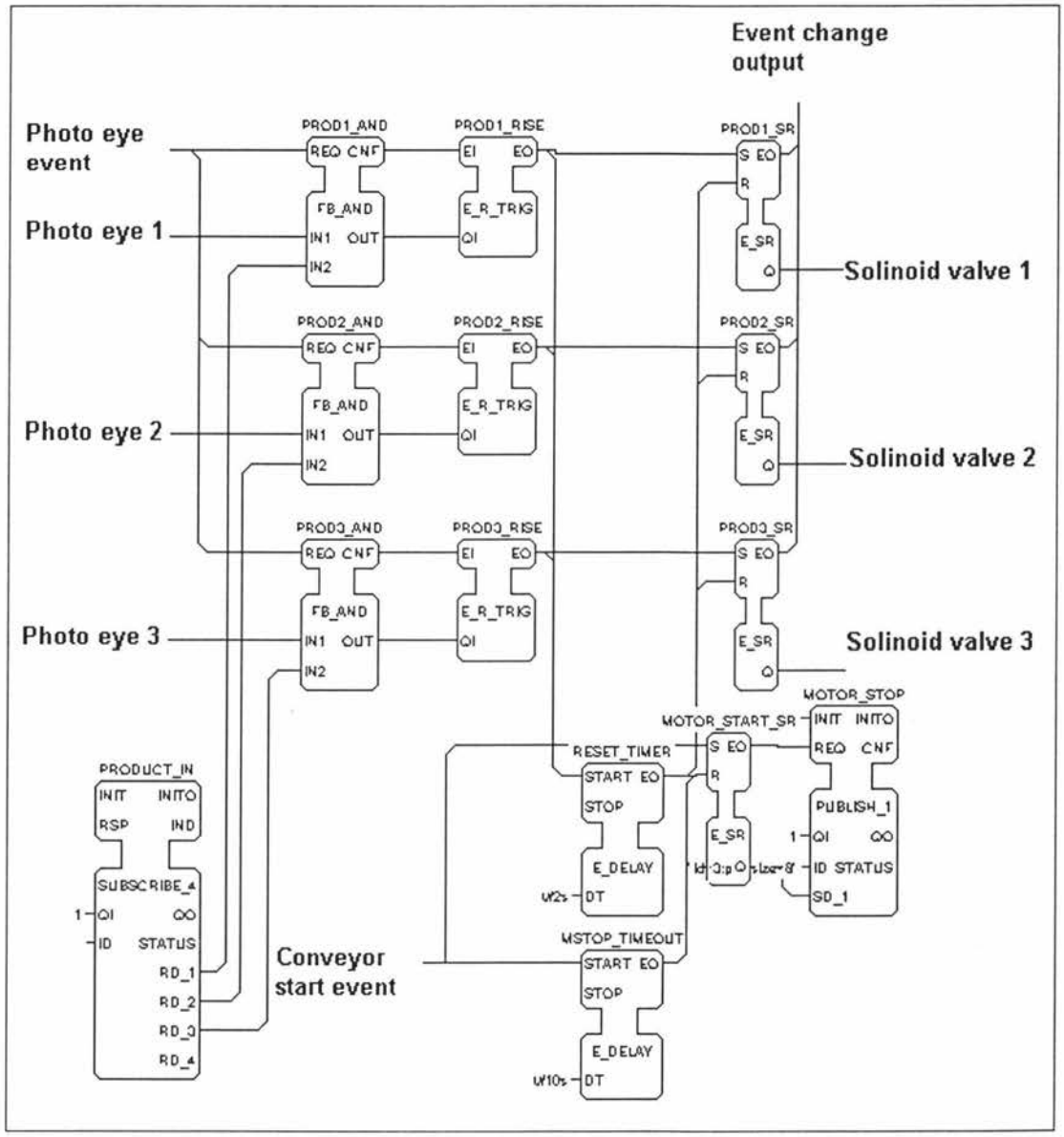


As this function block diagram is large it is explained in sections.

## Conveyor Control Section

The figure below shows the sorting software for the conveyor.

**Figure 6.21 Conveyor sorting control function blocks**



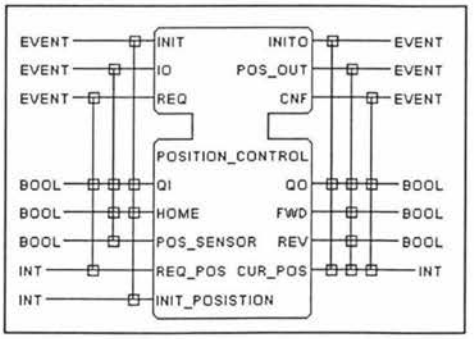

The SUBSCRIBE_4 function block is the DeviceNet interface from the bar code scanner. The outputs from this feed three FB_AND function blocks. The alternate inputs for the FB_AND function blocks are from discrete digital inputs that are

connected to photo sensors. Each FB_AND deals with a different photo sensor and air solenoid. The outputs from these are connected to E_R_TRIG function blocks. Each of these detects a rising edge from the photo sensor. The event outputs from the E_R_TRIGs are fed into three E_SR latches. The latch outputs are connected to the discrete digital outputs that control the air solenoids. The reset signal of the latches are controlled by an E_DELAY function block, which resets two seconds after the E_R_TRIG function blocks have detected a rise. The output of the E_DELAY function block also drives a set/reset latch function block (E_SR) which controls the conveyor motor. The output to the conveyor motor is an interface to DeviceNet. The second E_DELAY function block is the conveyor motor time out. The function block library that came with the IEC 61499 run-time was sufficient to control the product sorting requirements of the conveyor.

## Three Axis Robot Position Control

Developing the software for the control of the robot was more complicated than envisaged. The robot's position control was performed by three motors, each controlling a different axis with sensors to detect the position. The position sensors were reed and micro-switches placed on the axes of the robot. Some of the position sensors where wired together to the same I/O inputs so only a relative position could be given. To control which position the robot should travel to, a specialised counter function block was developed. This is shown in the figure below:

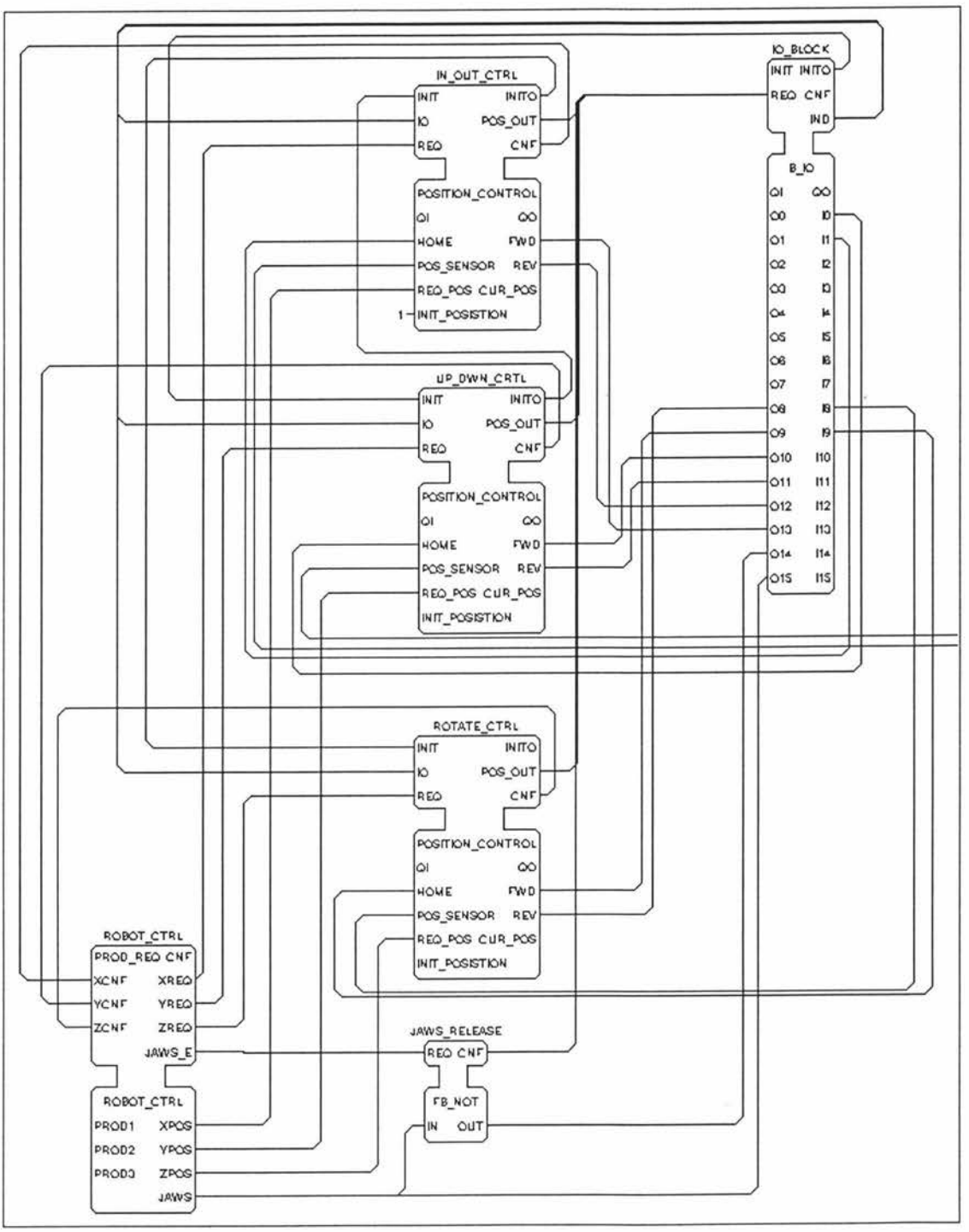
**Figure 6.22 Position control function block**



This is basically a counter with a compare function that indicates whether the current count is equal, over or under the requested count. The data inputs of the counter are; QI, HOME, POS_SENSOR, REQ_POS and INIT_POSISTION. The QI input is an event enable signal. The HOME input resets the count, a rising edge of POS_SENSOR increments the count with an I/O event. REQ_POS is the value of the target count value. INIT_POSITION is the default count value after initialisation.

On the occurrence of an INIT event, the REV output will be activated till the HOME position input is active. If the INIT_POSITION is not equal to zero then the FWD output will be activated until the current position equals the INIT_POSITION. Once the counter is at the correct position, the INITO event indicates that the device has initialised. The REQ_POS input latched with the REQ event will activate the FWD and REV outputs to tell the output device what it has to count to. When the current count value (CUR_POS) equals the REQ_POS then the FWD and REV outputs will go low, and the CNF event will be triggered indicating the counter is at the requested value. Events lines IO and POS_OUT are used to control the I/O device, setting the correct direction required to get the count to the right value. This function block was adequate for positional control, however when writing the code consideration was required to make sure that there was no thread deadlocking. Thread deadlocking occurs when multiple threads lock up waiting for each other to respond. What could have been considered with this function block was renaming the inputs and outputs so they are more suitable for a counter, as the name of this function block implies more positional control than a counter. In this case the counter is being used for position control. As shown in the figure below three of these function blocks were used, one for each axis. The HOME, POS_SENSOR, FWD, and REV were connected to the IO_BLOCK function block via decoding logic as required. The position requests are connected via other function blocks that are discussed further on.

**Figure 6.23 Robot control function blocks**




The above figure shows the three position control function blocks that control each axis. The ROBOT_CTRL function block handles the position requests. The original

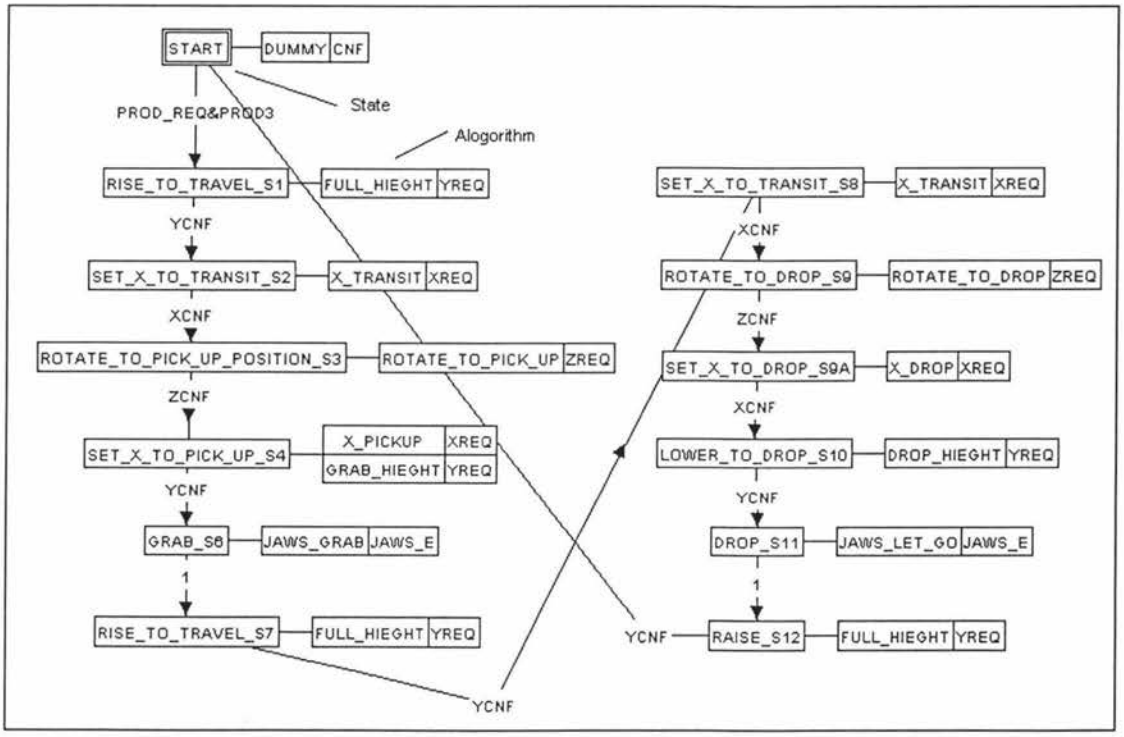
intention was to use a network of function blocks to control the robot, however this was not practical for the following reasons:

- The function block development kit software was having difficulty handling large function block diagrams.
- A large amount of time was required to program the device, thus making it impractical to write the function block network.
- It was doubted that the developer's CPU4 platform would handle the size of the application.

Thus a specialised function block was developed to handle the position requests. It also meant the programming code behind this function block could be shown at the EMEX show, as unlike most function blocks, the software is not intellectual property.

The ROBOT_CTRL function block consists of an 'Execution Control Chart' as shown in the figure below.

**Figure 6.24 Robot control execution control chart**

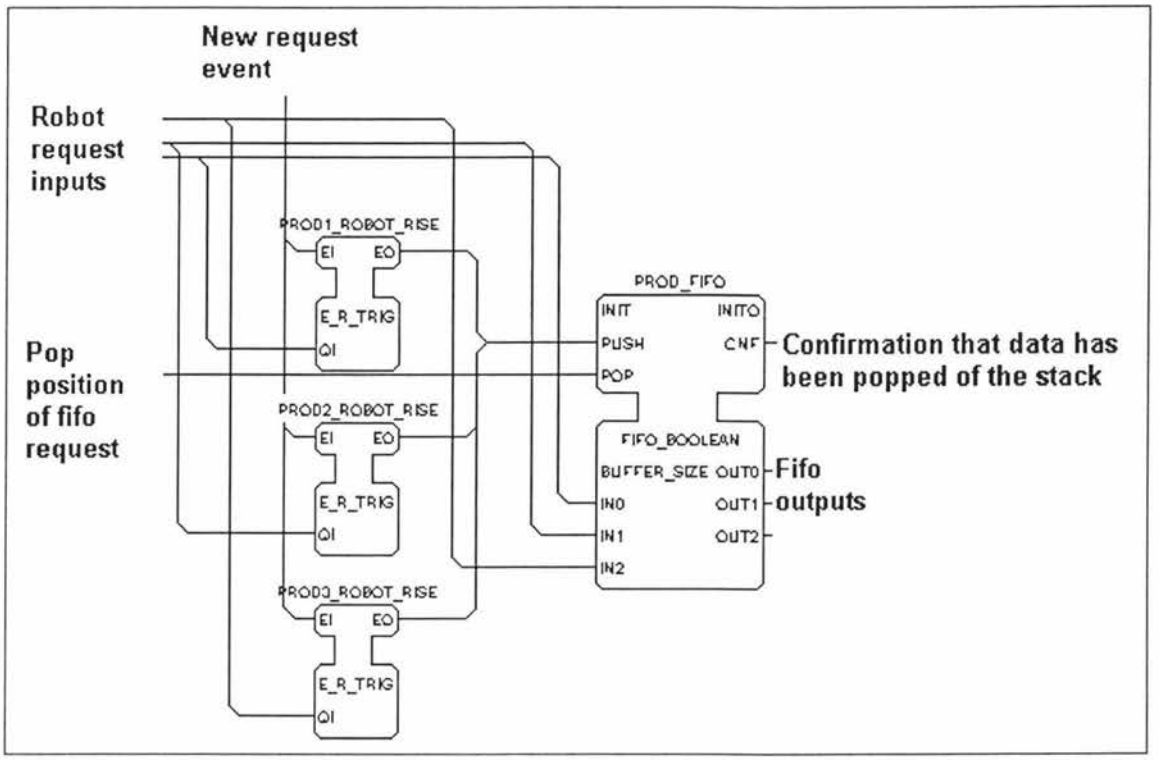


This is similar to a state diagram. When the function block is initialised the execution control chart is in the 'START' state. It is then controlled by the event flow. The arrows between states indicate the event that is required to allow the execution control chart to proceed to the next state. In the above chart, a PROD_REQ event will cause the chart to change state to the 'RISE_TO_TRAVEL_S1' state. The 'PROD_REQ' event is the event input that tells the robot that it has a product to pick up. The box to the side of each state is the algorithm. For the 'RISE_TO_TRAVEL_S1' state it is the 'FULL_HEIGHT' algorithm. This algorithm sets the data outputs to tell the vertical axis position controller to raise the robot to full height. The box attached to the side of the algorithm is the event output that the algorithm generates. Some of the algorithms check the data inputs so a decision can be made on what to set the outputs to. An example is the 'ROTATE_TO_PICK_UP' algorithm, this will check the three product data inputs, and rotate the robot to the appropriate pick up position based on what the data input was. Most of the events that allow a transition to the next state are 'XCNF', 'YCNF' and 'ZCNF'. These event inputs are connected to the POSITION_CONTROL function blocks and are confirmation that the robot has moved to that position. The making of a specialised function block for controlling the robot was fast and simple to develop, allowing the application to fit into the CPU4 target platform. However, with a class 1 IEC 61499 device there is no provision for creating new function block definitions, unless the user is the developer. It is understood at this stage that there are no class 2 devices that support this functionality.

## Robot Product Request Control

The position control of the robot is handled by the three POSITION_CONTROL and a ROBOT_CTRL function blocks. The selection of the appropriate pick up point for a product is determined by the activation of the photo sensors at the first two reject shoots and the pick up platform. The photo sensor input is a discrete digital input. As motion requests can occur faster than the robot is capable of responding, a buffer arrangement is required. The figure below shows the function block diagram to achieve this:

**Figure 6.25 Robot request function blocks**

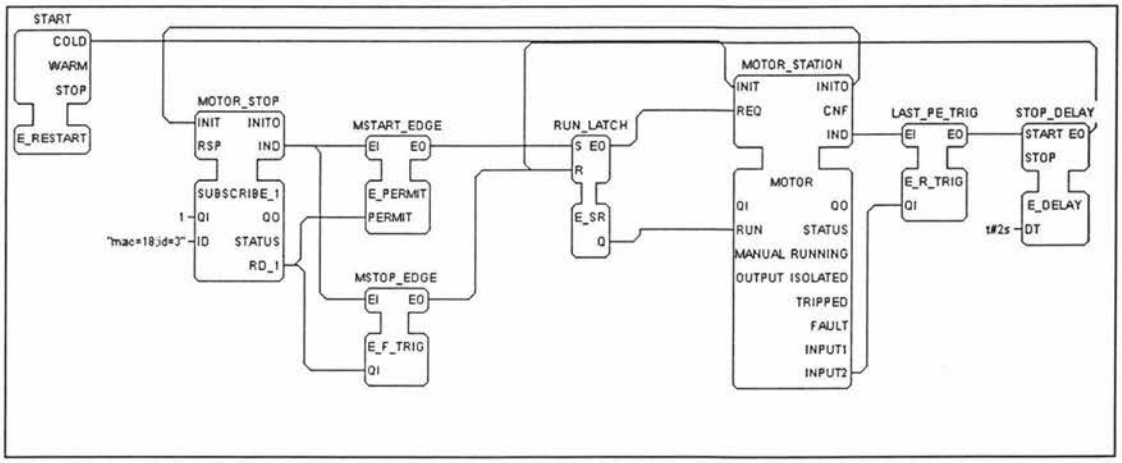



It consists of three 'E_R_TRIG' function blocks. These detect a rising edge of the request inputs that are fed into the PUSH events of the FIFO_BOOLEAN function block. The FIFO_BOOLEAN function is a First In First Out (FIFO) buffer for three boolean variables. The PUSH event puts data into the buffer. If the buffer is empty the input data is transferred to the output variables and a CNF event is generated. If the buffer already contains data, the input data will be stored. The POP event requests the next lot of data to be sent out of the buffer together with the occurrence of a CNF event. The POP event input is connected to the ROBOT_CTRL function block's CNF event output. The ROBOT_CTRL function block sends a CNF event after it has finished placing a block. The CNF output of the FIFO_BOOLEAN is connected to the PROD_REQ input of the ROBOT_CONTROL function block. The PROD_REQ input requests the robot to pick up another block. Development and use of a FIFO buffer became a method of being able to queue instructions to the robot.

## Conveyor Motor Control

A motor control station unit controlled the conveyor motor. It drives a relay contactor to switch on the conveyor motor, with some digital I/O. One of the digital inputs was connected to the last photo sensor on the conveyor which turns the conveyor motor off two seconds after a block has passed it. The function block application is shown in the figure below.

**Figure 6.26 Conveyor motor controller**




This shows a MOTOR function block that is the specialised interface to the motor control station hardware. The SUBSCRIBE_1 function block is the DeviceNet communications interface that tells the conveyor motor to start. There are two edge detection function blocks (E_F_TRIG and E_PERMIT) that are used to detect whether there is a request to start the conveyor or one to stop it. The E_SR acts as a set/reset latch to turn the conveyor motor on and off. The E_R_TRIG labelled LAST_PE_TRIG detects the rising edge of the last photo sensor on the conveyor. If the last photo sensor detects a product on the conveyor, a two second delay timer is started (E_DELAY) this is connected to the E_SR function block thus stopping the conveyor.

## Conclusions from Running this Application with IEC 61499 Function Blocks

The conveyor and robot application performed as it was intended, however it could be susceptible to mechanical failure and human interference. Some of these problems arose from the mechanical simplicity of the machine, the solution of which would require more sensors. Other problems related to the simplicity of the function block software written for this application. It was not written with any protection for the unplanned. The reasons for keeping the software simple was that it was easier to illustrate to people who have no understanding of this technology, and to keep within the limits imposed by the current hardware and development software. A further improvement that could have been considered for demonstration of a distributed system was to use another 16 way I/O module so the conveyor control could be on a separate node from the robot controller.

The success of having the application running without a centralised processor clearly showed that IEC 61499 technology is suitable for automation. Observation of the application's behaviour demonstrated some of the advantages of having a distributed control system, some of which are listed below:

- Reduced single point of failure: Removing the conveyor motor control station from the control network showed that the robot and bar code scanner were still capable of operating.
- Reduced use of network bandwidth: Using a DeviceNet interface card attached to a PC monitoring the packets sent over DeviceNet showed that two lots of 8 byte messages were sent over DeviceNet at an average of every five seconds. For a PLC based system it may be possible to have similar length messages, however all I/O activity would be sent to the central controller, whereas with the distributed control application most of the I/O activity was confined within the device itself. If all I/O activity was sent on the network, high use of network bandwidth would be required.

- Standardised hardware and software: The devices used in the test applications consisted of standard hardware and software modules, with the differences of the I/O devices attached to them. The advantage of this, is reduced costs of manufacturing for low volume products.

## Evaluation of the IEC 61499 Standard at Present

The test applications that are discussed in this chapter clearly show that the use of the IEC 61499 standard is feasible with embedded microprocessor platforms. The previous discussions have illustrated the advantages and disadvantages particular to applications. This section discusses the overall problems and issues that need rectifying before the technology will be commercially acceptable.

### Function Block Programming Software

At commencement of this research there were two IEC 61499 software editors; FBDK from Rockwell Automation, and CORFU FBDK, distributed by the University of Patras, Greece. The CORFU software appeared to have a relatively friendly user interface and was the closest to a commercially acceptable package. This application was not chosen because while the software could draw function block diagrams it could not be interfaced with any function block run-time environments.

The FBDK from Rockwell Automation was the software package used throughout this research, while the software was difficult to use and has a number of software bugs, it was capable of performing all that was required for this research. This was developed by Dr Christensen to assist his research with the IEC 61499 standard. The software licence agreement states; "This is experimental prototype software which may not be offered in future as a commercial product. This software is not intended to be used for the design or implementation of actual control systems. Any liability resulting from such use is the responsibility of the user" (FBDK, 2003). A discussion with Christensen showed that he is unsure to how this software will be developed in the future. Use of

this software showed that with small applications it was useable. When developing the larger application for the conveyor and robot control the software had difficulty managing the screen layout making it hard to write the application. The software has limited drag and drop facilities making it harder for the software developer to use. Christensen did mention another function block editor; "There has been some development with a VISIO based function block editor, however due to decisions made by the company developing this software, the company ceased development of this package." (J. H. Christensen, personal communication, September-October, 2003). With the limitations of the development software, a more reliable and user friendly software package needs developing.

## Hardware Development

As discussed throughout this thesis there have been limitations with the developer's hardware platform. While their platform has been developed into a commercially acceptable form factor, the small amount of memory and slow processor speed limit the size and use of applications. Discussions with the developer have indicated that they intend to upgrade to a more powerful hardware platform.

The test applications clearly showed the use of the same hardware platform on a range of three devices; a serial communication interface, a motor control station, and an I/O module. All devices consisted of the core CPU4 module with the function block run-time software. The hardware differences were the I/O expansion devices attached to it.

## IEC 61499 Function Block Library

The function block library developed for the function block run-time environment was limited to what the author required for this research project. For commercial use of this standard, there needs to be interaction with product end users, to find out what function blocks they need. With the development of the robot application it was noticed that there could be some more sequential control function blocks. As all the devices belong

to class 1, the only way of expanding the function block library is to upgrade the product firmware. Firmware upgrading is often not available to the end user.

# Chapter 7 Upgrading DeviceNet for use with the IEC61499 Standard

DeviceNet is a communications standard based around the CAN network architecture used for distributed I/O centralised control systems. There are two types of communication messages sent. I/O messaging is used for sending I/O data, and explicit messaging is used for configuration of devices and initiating the I/O messages. Typically a distributed I/O network will consist of a central PLC scanner which will poll the field devices with I/O messages. Prior to development work with the IEC 61499 standard, the developer company upgraded their DeviceNet communications software to facilitate the use of the IEC 61499 standard, by adding peer-to-peer I/O messaging. The details of this upgrade are published in "Distributed Functionality for DeviceNet Communication" (Meek, 2003b). When the developer company performed this upgrade, the requirement for distributed control and the demands placed on DeviceNet by the implementation of the IEC 61499 standard were unknown. This resulted in the upgrade being performed to fit their perception of the IEC 61499 standard at the time. The upgrade consisted of the addition of peer-to-peer communications and rebuilding of the DeviceNet stack to application software interface.

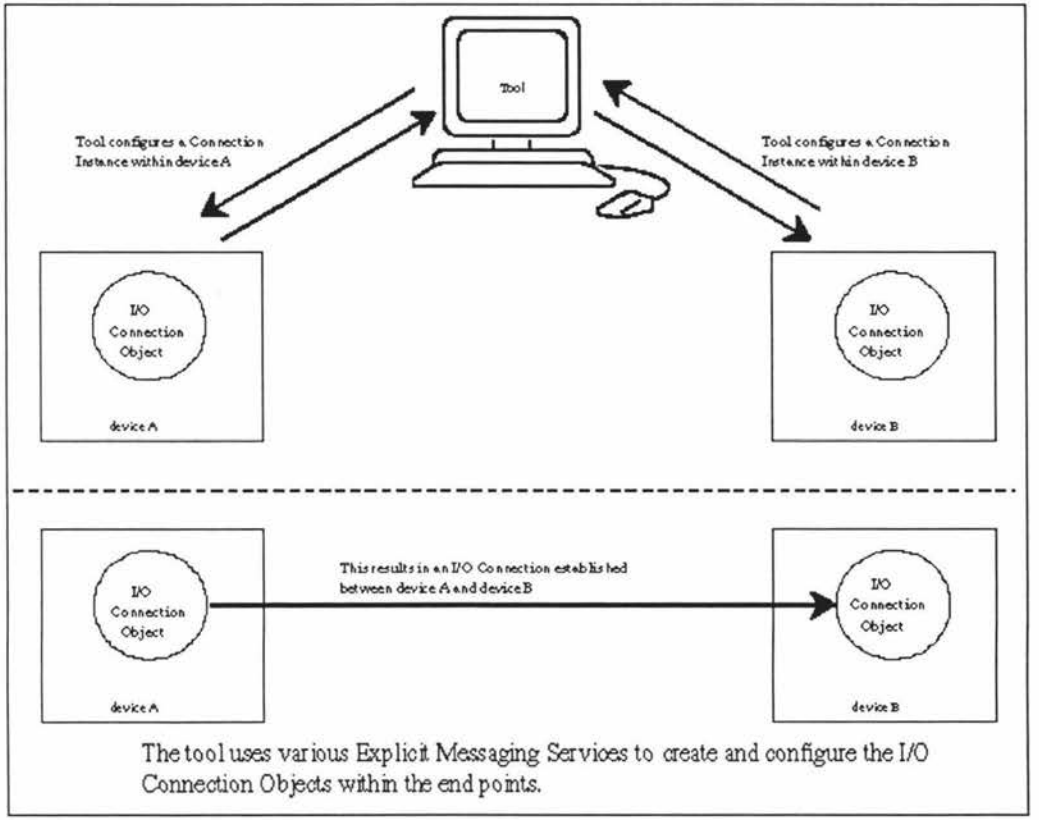
## Problems with the Existing DeviceNet stack for use with IEC 61499 Run-time Environment

When first trying to use DeviceNet with function blocks there were some complications, such as packet encoding formats and peer-to-peer connection establishment. The communication interface in the DeviceNet stack was built as required by the ODVA DeviceNet Specification, although the standard is not strict in this area. The DeviceNet communication protocol is object orientated, where each I/O connection is limited to attributes (members) of an application object, or a group of attributes that are arranged into an array of bytes known as an assembly. This results in a typical I/O message

following a set format that is defined by the device manufacturer, with no encoding of different data types. The IEC 61499 standard defines an encoding format that consists of an array of bytes that is sent over a communications interface, starting with a data type identifier, followed by the data associated with that data type. The other issue was that the DeviceNet stack required creation of all the application objects before initialisation, instead of dynamic creation of application objects. Dynamic creation of objects is a requirement with IEC 61499 programs because the DeviceNet stack has to be fully functional while downloading the function block programs. This resulted in an upgrade of the DeviceNet stack's application interface.

The peer-to-peer I/O messaging communications of the developer's DeviceNet stack was fully functional apart from minor bug fixes. However at the time the DeviceNet stack was upgraded, the developer and the author envisaged the creation of the peer-to-peer connections between devices be performed by a management node on the network. The figure below shows an example of this.

**Figure 7.1 Peer-to-peer connections being created with a management node**




(Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. 3-12)

This resulted in the DeviceNet communication software not having the ability of creating peer-to-peer connections between nodes by themselves. With the IEC 61499 standard on communication function blocks, this architecture was not suitable, and it was decided that the DeviceNet stack needed the functionality to create connections autonomously. This functionality is termed 'explicit messaging capabilities'.

## Upgrading the Application Interface and Developing the Packet Formats

The IEC 61499 standard and FBRT were built around ethernet communications, with the data packet lengths differing depending on how much data is sent. With DeviceNet the I/O messaging packet lengths are predefined by the application and the manufacturer. With function blocks it can be hard to determine the length of a packet during configuration, particularly if sending variable length strings. The function block run-time environment source code uses the standard Java ethernet communications API (a term used for a library of methods with Java), that after configuration has three basic methods to send and receive data. These are 'in()', 'out(data)' and 'flush().' Calling the method 'in' returns data that is in the receive buffer, 'out' places data into the transmit buffer and flush sends the output data. With Java being a transportable language between platforms it was decided that the DeviceNet interface should use the same in(), out(data) and flush() methods.

Considering the variable data lengths, a DeviceNet Java API that is used in a similar way to the Ethernet API, was built. The standard method of attaching an application to DeviceNet communications is to create an object of a specified DeviceNet application class. When a DeviceNet communications link is created the communications link will communicate to an attribute (member) of that DeviceNet application object. To create a communications interface function block, a DeviceNet application object needs creating.

## Creating a DeviceNet Application Object

A typical DeviceNet application class has a unique identifier that is either assigned by ODVA (Open Device Vendors Association, the DeviceNet standard governing body), or by the product vendor if ODVA have not specified a class for the application.
ODVA have assigned the identifiers as listed in the table below:

**Table 7.1 Class identifiers**

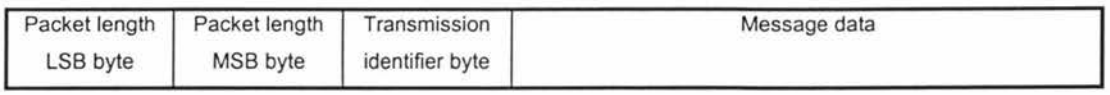
| Type | Range | Quantity |
|---|---|---|
| Publicly Defined | $00_{hex}$ - $63_{hex}$ | 100 |
| Vendor Specific | $64_{hex}$ - $C7_{hex}$ | 100 |
| Reserved | $C8_{hex}$ - $FF_{hex}$ | 56 |
| Publicly Defined | $100_{hex}$ - $2FF_{hex}$ | 512 |
| Vendor Specific | $300_{hex}$ - $4FF_{hex}$ | 512 |
| Reserved | $500_{hex}$ - $FFFF_{hex}$ | 64,256 |

(Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 2, p 2-22)
The identifiers they assign are from 1h to 63h, and above 63h are vendor specific. With no IEC 61499 device profile and class defined by ODVA the "IEC 61499 interface class," which will be the term used for the application class for the DeviceNet communication interface, was assigned the value of 70h. 70h was the chosen identifier value, because it was in the vendor specific range of identifiers and it allows space for specific application objects which are used by the DeviceNet stack which start at 64h.

For an identifier that defines where a function block should send data to, it was decided that the user of the communications function block will use the remote node's "MAC identifier" (the DeviceNet nodes a unique identifier) and the identifier of the IEC 61499 interface object (an object of the IEC 61499 interface class). Therefore when the user initialises a communications interface function block, an object of the IEC 61499 interface class is created. The attributes and details of its communication interface are discussed appendix G.

A typical DeviceNet application class has a number of predefined members to it with predefined data types. With the IEC 61499 standard the data types are unknown until the packet is sent. This means that predefined data types are impractical using the IEC 61499 standard. This was solved by creating attributes (members) of the DeviceNet array data type that contain the production and consumption data (production and consumption are the terms used for transmitting and receiving information with DeviceNet). These attributes are the communication data from the data inputs and outputs of the communication interface function blocks. When the function block communication interface is requested to send data, the out(data) method is called, this places data in the production data attribute forming the data packet shown in the diagram below.

**Figure 7.2 Device function block packet format**




The 'out' method places data into the 'message data' section of the production data attribute and sets the packet length. When the flush routine is called the transmission identifier byte is incremented, and the DeviceNet stack is triggered for message production. Message production triggering is discussed further on. When the DeviceNet stack produces a message, the consuming nodes will have this message placed in the consumption data attribute of its IEC 61499 interface object. The 'in' method will pass the data from the consumption attribute to the communication interface function block in the consuming node.

This method of sending data works, however the packet length sent over DeviceNet is a predefined length with the data length bytes in the packet only specifying the number of bytes that are used in the packet. This unfortunately means the user has to estimate the number of bytes they are going to send and receive when the communication interface function block is configured. If the user does not set the packet size large enough the

communications interface will not function correctly, whereas when selecting too large a packet size the interface will function correctly, but will use more network bandwidth. This problem has arisen because of differences between two standard setting bodies. This means the interface will have to remain like this until an official communication interface has been set by the standard setting bodies. It is proposed that once the developer and author have DeviceNet with IEC 61499 function blocks working correctly, a specification will be submitted to the standard setting bodies (ODVA for DeviceNet and WG6 for IEC 61499). A draft version of the DeviceNet with IEC 61499 specification is included in appendix F.

The communication interface function blocks need configuration parameters passed to them. With ethernet this is the IP address and interface port. For DeviceNet it was decided to use the MAC and IEC 61499 interface object identifers. Two further parameters are added to define consumption and production messages sizes. These define the packet lengths that were discussed in the previous paragraph. Adding the consumption and production size parameters to the configuration string means that the same communication interface function blocks can be used as with ethernet.

Once the IEC 61499 interface object has been initialised, the communication link needs creating. There are two types of messaging used; client/server which is bi-directional communication between two nodes and publisher/subscriber communications which is multi-cast between multiple nodes. A client/server connection requires the server node to create the interface object. The client node has to create an explicit messaging connection to extract the consumption and production sizes from the server node. From this information the client will create its own interface object and create the communications link between the two nodes. The publish/subscriber connection requires the publisher node to create its own interface object and create a connection to produce data for itself. The subscriber node will create an explicit messaging connection (discussed in the next section) with the publisher node. This is to extract its

production data size and connection properties to create its own interface object and communications connection.

## Peer-to-peer Communications with DeviceNet

Prior to commencement of this project, the developer company upgraded their DeviceNet stack to incorporate peer-to-peer communications. The peer-to-peer communication protocol has been developed to the requirements of the ODVA DeviceNet standard under the section of Dynamic I/O. The peer-to-peer I/O configuration consists of creating an explicit messaging connection for configuration of peer-to-peer I/O connections. At commencement of this project the developer's DeviceNet stack could not create explicit messaging connections with other nodes. To achieve this the DeviceNet stack underwent a further upgrade.

## Creating Explicit Messaging Connections

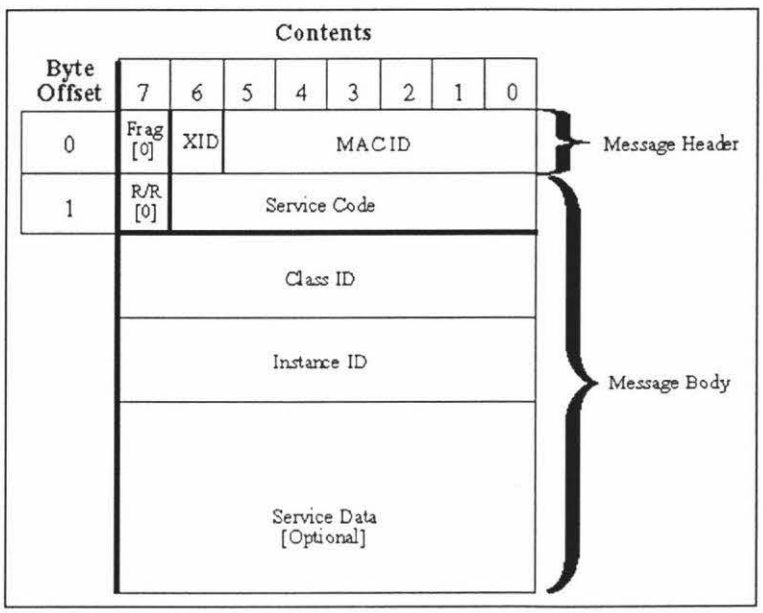
Explicit messaging connections are used to send and receive configuration and device information. To create an explicit messaging connection the client device will send a UCMM (Unconnected Message Manager) message. The UCMM handles the creation of explicit messaging connections. This message asks the server device to create an explicit messaging connection with it. The previously upgraded DeviceNet stack could only receive UCMM and explicit request messages. The developer's DeviceNet stack is coded in C and uses state machines to handle the multiple communication connections. The I/O messaging connection is simple as there is only one end point. To upgrade the stack to send UCMM and explicit messages, Java was chosen as the software language. Using Java simplified the code as each explicit messaging connection is another object, whereas writing the code in C would have meant modifying the DeviceNet stack's state machine. When modifying the state machine it is very easy to introduce bugs.

When the server device receives a UCMM request message it will create an explicit message connection, then send a UCMM response message with the details of the explicit message connection. Refer to appendix I for explicit message formats.

## Sending Explicit Messages

An explicit message frame holds identifier information in the format shown in the figure below.

**Figure 7.3 Explicit message packet format**




(Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. 4-19)

The first byte of an explicit message is the MAC identifier, the identifier of the node receiving the message. The 'Service Code' byte is the instruction of what the receiving node is to do with the message. Typical service codes are either setting or getting attribute information. The class and instance identifiers define the destination object. Typically with the function block run-time environment the only class identifiers which are accessed is the IEC 61499 interface class and the connection class. The IEC 61499 interface class is used to find out information on what type of connection is to be created. The connection class is used to create and configure the peer-to-peer I/O connection. The optional service data typically contains an attribute identifier of what attribute data is going to be modified or read. If attribute data is modified then the new attribute data follows the attribute identifier. More details of the explicit messaging

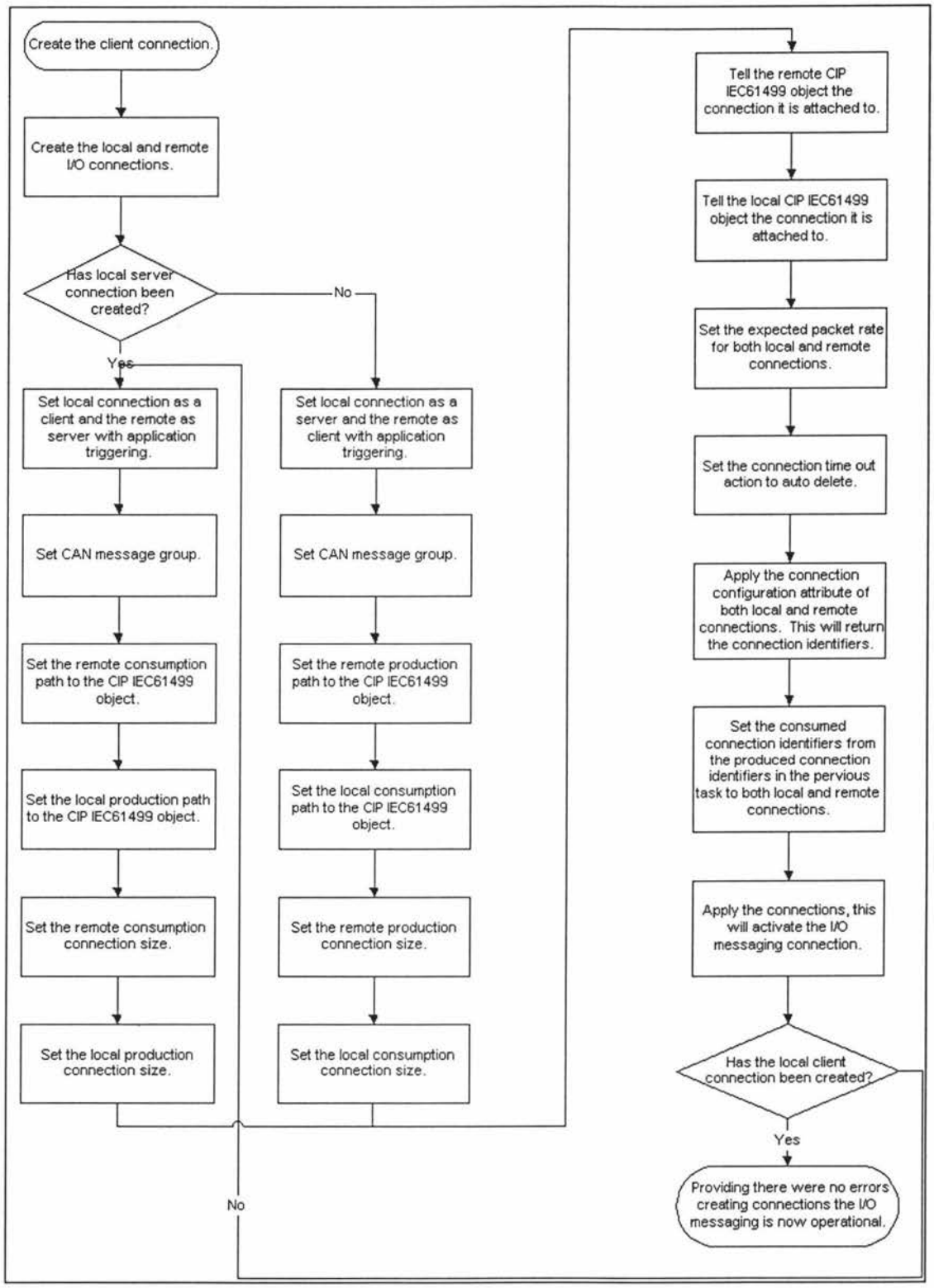
format are explained in appendix I. The sequence of the configuration of explicit messages is illustrated in figures 7.4 and 7.5.

## Creating I/O Connections

I/O connections are created using explicit messaging to access the connection class, thus creating a connection object. This procedure is flow charted in figures 7.4 and 7.5. When creating an I/O connection the DeviceNet specification defines three ways of triggering data production: cyclic, change of state, and application. All methods of triggering have a cyclic component, which makes sure that a message is sent after a defined period of time. It is unclear in the DeviceNet specification whether a complete packet has to be sent for a cyclic heartbeat, however packet monitoring of other manufacturer's products shows they send a complete packet. Use of cyclic triggering requires a fast cycle rate if a quick response is required. The fast cycle rate means that more network bandwidth is used. Change of state triggering still has a cyclic component which acts as a heart beat message. When data is produced as stated in the specification "Production occurs when a change-of-state is detected by the Application Object." (Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1 p. 5-10). Inside the developer's communication software, two buffers are continually compared to detect a change-of-state. This method of triggering was not chosen, as application triggering was considered better. As IEC 61499 function blocks are event driven, application triggering was chosen. The difference between application triggering and change of state is as specified; "The Application Object decides when to trigger the production." (Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1 p. 5-10). Inside the application code a manual trigger is passed to the DeviceNet communications software trigger data production. The advantage of using application triggering over change of state, is because change of state requires the DeviceNet stack to continually compare two buffers to see if there is a change of state. This requires more use of processor time and memory. There were no problems using application triggering.

For client/server connections, two connections are used, one for each direction. The reason for this is that DeviceNet I/O connections are client/server based, with the server connection only able to produce data when the client has requested it. To solve this problem two connections are used; each device has a client and a server connection. The client connections are used for producing data and the server connections are used for consuming data. The figure below is an overview on how I/O connections are created.

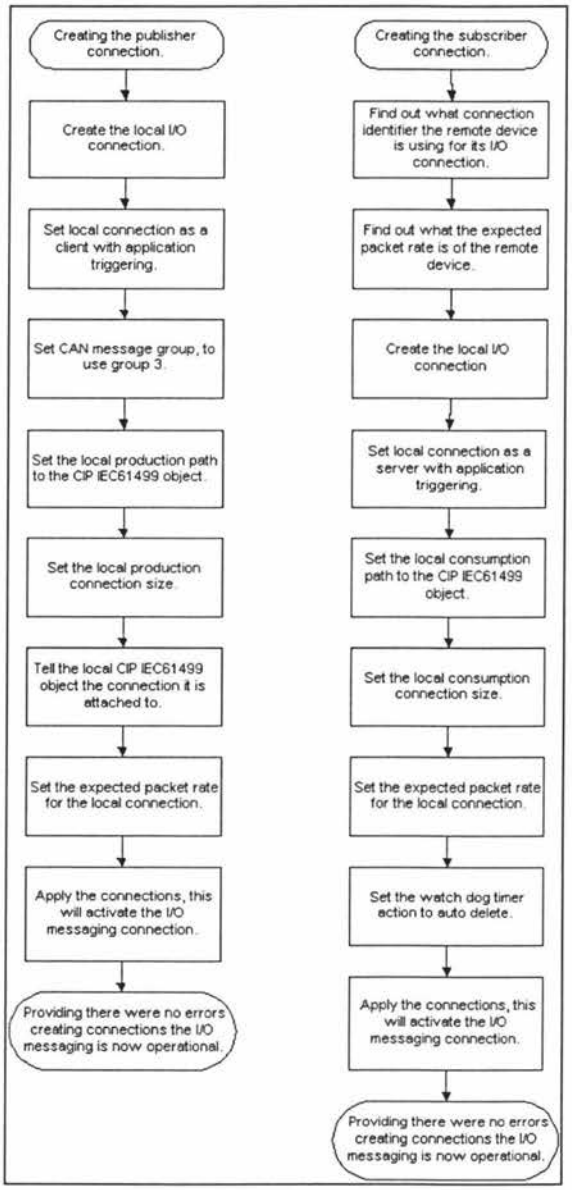
**Figure 7.4 Client Connection Establishment**




The specific connection object attributes are defined in appendix J.

Publisher/subscriber communications used I/O messaging connections similar to client/server communications, except data transfer is uni-directional (publisher to subscriber). With DeviceNet being a producer/consumer communications protocol, all producing connections are multicast. Point-to-point connections are also multicast however only one node is configured to consume the data. The figure below illustrates how the publisher/subscriber connections are configured.

**Figure 7.5 Publisher/subscriber connection establishment**



The publisher device creates an I/O connection for itself, however when it creates this connection it does not communicate with any nodes. The connection object attributes are set the same way as the client/server connections, with the exception that only a producing connection is configured using CAN message group three. The subscriber device will create an explicit messaging connection to the publisher device and communicate to the interface object to find out what connection instance is associated with the publisher's connection. The connection object of the publisher device is checked, so the subscriber device knows what multi-cast packets are assigned to that multi-cast communication link. The other attributes of the connection object are the same as the client/server communications. With DeviceNet being a producer/consumer communication system, the publisher/subscriber connections worked well.

## Configuration of IEC 61499 Devices

When the developer company received the third party IEC 61499 run-time environment, all communications were ethernet based, including configuration of devices. When using ethernet no specialised programming tools are required, as most PC's have ethernet interface cards in them. However this is not the case with DeviceNet. There are DeviceNet interface cards available, however not within the developer's price range. A decision was made to make a serial to DeviceNet adaptor for PC communications. The reason for using serial communications was to take advantage of the developer company's hardware.

The function block development software communicates with a function block run-time environment that runs on the PC, acting as a communications adaptor. The ethernet communication function blocks on the PC were modified so they could communicate with the serial port to use the serial to DeviceNet communications adaptor. The serial communications protocol consists of a packet format shown in the figure below.

**Figure 7.6 Serial communications packet**

| Command byte | DeviceNet MAC id byte | DeviceNet IEC 61499 object identifier byte | Message data, using the standard client/server packet format |
|---|---|---|---|

There are four commands:

- Create a DeviceNet connection.
- Send data.
- Receive data.
- Delete DeviceNet connection.

When a 'create' command is sent, a developer connection is created in the serial to DeviceNet interface device. The 'MAC identifier' and 'identifier byte' contain information about the communication end point, which is the node that is being configured. The send and receive commands are for sending and receiving of configuration data. The delete command closes the client/server connection and the interface object within the serial to DeviceNet interface.

Tests of this interface for configuration showed that it was functional and simple. However, debugging the interface was challenging, because three devices and two communication protocols required fault finding simultaneously. As discussed with the test applications of IEC 61499 in chapter 6, the communication interfaces worked well. As part of trying to maintain IEC 61499 standard, a DeviceNet profile has been written and is included in appendix F.

# Chapter 8 Conclusions

The use of a system based on IEC 61499 function blocks for the demonstration machine (robot and conveyor application) for the EMEX show Auckland, May 2004, has proved and shown that the IEC 61499 function block programming system can be used with embedded microprocessors to control real-time applications. Observation of the robot and conveyor test application has shown the advantages of the IEC 61499 function block programming system as well as where improvements are required for commercialisation.

Using function block programming with DeviceNet has shown that the technology has reduced network bandwidth. It has shown that DeviceNet is compatible with IEC 61499, with an acceptable level of performance.

The not so user friendly and unreliable function block programming tools could prevent commercialisation. At present all function block development tools that the author has used have been designed to assist researchers who accept the software limitations. However there is doubt that end users will accept this.

## Project Review

Two big decisions were made in this project. This was choosing Java instead of C for function block development and using Rockwell Automation's IEC 61499 run-time environment. The decision to use Java instead of C allowed the developer company to adapt an IEC 61499 run-time environment for their needs, thus saving development time. Had the developer used C as the programming language, with its associated better performance, it would have required more development time and scope for further improvement would have been limited. The porting of the KVM (Sun Microsystems embedded Java virtual machine) proved to be a large task, however, it has allowed the developer to customise the virtual machine and run with existing legacy software. The

performance and reliability of the virtual machine has exceeded expectations. Since porting the software and after some initial minor bug fixes, the virtual machine has caused no problems.

Choosing Java instead of C lead to complications with the developer's CPU4 platform with its 8 MIPS microprocessor. At the time the decision was made to use Java, consideration was required on what the most suitable hardware platform was. The decision was made to continue development with developer's CPU4 platform although the limited amount of memory and processor speed would constrain the size of the application. The performance of the application showed that the CPU4 platform had its limitations.

The developer's CPU4 platform with its 8 MIPS microprocessor could be used for the development of a pilot distributed control system using function block programming conforming to the IEC 61499 standard. The limited amount of memory available on the platform constrained the size of the application. The limited capability of the processor also resulted in poor response to events in the system. This platform performed adequately for the robot and conveyor application, however it was noticed that the robot position control was only just within acceptable tolerances with the I/O response time. Using this platform has shown the developer company what will be required of the new hardware platform. It is their intention to use a 160 MIPS microprocessor, and the author's estimated memory requirement is 4 Mbytes of ROM and 2 Mbytes of RAM using the current DeviceNet technology. If ethernet communication is used, more may be required.

The decision to use Rockwell Automation's IEC 61499 run-time environment for the developer's platform has given the developer a head start of about a year to commercialisation of this technology. Use of the third party run-time software has improved the skill base of the author and developer, with greater understanding of function block programming techniques. The performance of the run-time system has

indicated that with some more development in areas of the programming interface, and an improved function block library, the developer could have the IEC 61499 function block programming system commercially viable in a relatively short time frame. To expand the function block library, market research is required to establish what function blocks are required to meet the needs of clients.

# Appendix A IEC 61499 Device Class Definitions

This appendix presents the different classes of IEC 61499 devices. The table below lists the commands and services that are available for each class of device.

**Table A.1 Device class definitions**

| CMD [a] | OBJECT | CLASS 0 | CLASS 1 | CLASS 2 |
|---|---|---|---|---|
| CREATE | type_declaration | | | Required |
| | fb_type_declaration | | | Required |
| | fb_instance_definition | | Required | Required |
| | connection_definition | Required [b] | Required | Required |
| | access_path_declaration | | Required [c] | Required [c] |
| DELETE | data_type_name | | | Required |
| | fb_type_name | | | Required |
| | fb_instance_reference | | Required | Required |
| | connection_definition | | Required | Required |
| | access_path_name | | Required [c] | Required [c] |
| START | fb_instance_reference | Required | Required | Required |
| | Application_name | Required | Required | Required |
| STOP | fb_instance_reference | Required | Required | Required |
| | Application_name | Required | Required | Required |
| KILL | fb_instance_reference | | Required | Required |
| QUERY | all_data_types | Required | Required | Required |
| | all_fb_types | Required | Required | Required |
| | data_type_name | | | Required |
| | fb_type_name | | | Required |
| | fb_instance_reference | | Required | Required |
| | connection_start_point | | Required | Required |
| | Application_name | | Required | Required |
| | access_path_name | | Required [c] | Required [c] |
| READ | access_path_name | Required [c] | Required [c] | Required [c] |
| WRITE | access_path_data | Required [c] | Required [c] | Required [c] |

[a] See 3.3.2 for definition of the semantics of these commands

[b] Only connection of a new *parameter* value to a *data input* of a function block is required of Class 0 devices.

[c] This capability is required only in devices that support *access paths*.

(IEC T65/WG6(PT4PAS)FD, 2002, p. 8)

Definitions of commands are listed below:

## Create

This creates function blocks, data and event links.

## Delete

This is used for deleting function blocks, events and data links.

## Start

This is used for starting applications and function blocks.

## Stop

This is used to stop function blocks and applications.

## Kill

This makes specified objects, un-runnable.

## Query

This is used for querying the status of a function block applications.

## Read

This reads data from communication links.

## Write

This is used for writing data to communication links.

# Appendix B SICK Barcode Scanner Interface Technical Manual

This is an extract out of the TCS-DNSI-SICK technical manual that is the product that was used in the Java evaluation tests.

(TCS, n.d., p. 1, p. 5-14, p. 22):

# Tait Control Systems

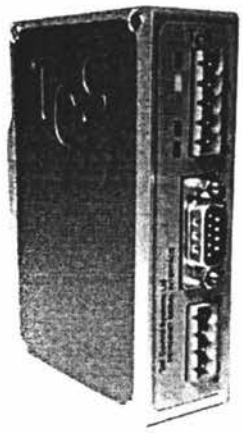
*Innovators in Industrial Control Equipment*

## TCS DeviceNet Serial Module

DeviceNet Interface Technical Manual

## Cat No. TCS-DNSI-SICK

Rev: 4.1



Head Office: 34 The Boulevard

P.O.Box 20-489

Te Rapa

Hamilton, New Zealand

## Getting Started

The TCS-DNSI-SICK module enables the connection of a SICK bar code scanner (the Serial Device) to a DeviceNet network. The TCS module will require DeviceNet node parameters configured. The Serial Device will need its communications parameters set. The PLC connected to DeviceNet, will need programming to handshake with the TCS Module.

Note: - Throughout this manual are references to 'DeviceNet Manager' programming software. Equally 'RS Networx' can be used.

### Step 1:

Wire and commission the TCS module in accordance with the mounting and connection detail. Refer to *Hardware Assembly* section.

### Step 2:

Power up the TCS module and network. The DeviceNet Manager should now detect the TCS module. Check that the correct network and module status LED indicators are displayed. Refer to *Display Indication* section.

### Step 3:

The Mac ID and DeviceNet Baud Rate can be configured using the DIP switches accessible on the underside of the TCS module. By performing an online build, and uploading the EDS (Electronic Data Sheet) from the devices parameters can be set to match your application.

### Step 4:

Configure the SICK Barcode scanner communications protocol to match the default interface protocol. Refer to *Setup the Barcode scanner* section.

### Step 5:

Connect the SICK barcode scanner to the DeviceNet interface module as per the **Wiring Details** section.

Note: Use only the correct plugs. Ensure that all terminations are correct prior to applying power. Be sure that the RS-232 and I/O connections are not mixed and that no shorts between pins exist.

### Step 6:

Determine the maximum barcode size and set the value into the module parameter 8.

### Step 7:

Add 2 to the number of characters and this becomes the module production size. Map the device into the barcode scanner (RXD 2 + n characters bytes, TXD 2 bytes) as a polled or change of state/cyclic connection.

Note: The factory default configuration sets the module for discreet trigger input. To trigger the module for PLC trigger you need to set bit 4 of the function word and then by toggling bit 0 of the function word the module will send the trigger command to the barcode scanner.

Providing you have set all the parameters correctly in both the barcode scanner and the DeviceNet interface module you should now be able to aim the scanner at a valid barcode and have the data returned to the PLC data table location as mapped.

Refer to the trouble shooting section for tips on fault finding.

## DeviceNet Information

The TCS module allows the SICK bar code scanner to operate as a slave device on the DeviceNet network. The TCS module supports the predefined master/slave connection set's, polled, change of state/cyclic messaging.

Each TCS module has different numbers and types of parameters. Some of these will be user configurable, and some read-only. Each user configurable parameter may be edited in the same way.

Upload parameter information from the TCS module by adding it to the DeviceNet network using DeviceNet Manager or equivalent, then double click on the device icon, or click the right hand mouse button and choose the Configure Device option.

## DIP Switch Settings – DeviceNet MACID and Baudrate

The TCS module has an eight way DIP switch accessible on the underside of the module. These DIP switches set the DeviceNet MACID (i.e. the node number) and the DeviceNet Baudrate.

### MACID

Each device connected to DeviceNet must be given a unique MACID. DIP switches 1 to 6 set the MACID. Switch 6 is the MSB and switch one is the LSB. The range of the MACID is 0 - 63.

### DeviceNet Baudrate

Each device connected to DeviceNet must know the present Baudrate of the network. DIP switches 7 and 8 set the DeviceNet Baudrate.

| DIP Switch 7 | DIP Switch 8 | DeviceNet Baudrate |
|---|---|---|
| OFF | OFF | 125kBaud |
| ON | OFF | 250kBaud |
| OFF | ON | 125kBaud |
| ON | ON | Software Set (unused) |

## Connection Size

Rxd Size: 2+Bar code size bytes

Txd Size: 2 bytes

Note: This is Rxd and Txd from the scanners perspective.

## Transmit Data (Output from PLC/scanner, also known as Consumption Data):

| Element | Description | Size (words) |
|---|---|---|
| 0 | Function Word ( see Status Word bit definition table below ) | 1 |

## Function Word Bit Definition

The *Function Word* come from the PLC to the TCS module and the Serial Device.

| Bit | Description |
|---|---|
| 0 | DEVICENET_READ_TRIGGER |
| 1 | READ_CYCLE_COMPLETE_ACK |
| 2, 3 | (Reserved) |
| 4 | TRIGGER_SELECT |
| 5 – 7 | (Reserved) |
| 8 | DISCRETE_OUTPUT |
| 11 – 15 | (Reserved) |

### Bit Definitions

DEVICENET_READ_TRIGGER

This bit is used to trigger a barcode scanner read. This bit is only valid when both the barcode scanner is configured to receive a serial trigger command and when the TRIGGER_SELECT is set to 1 (DeviceNet trigger).

READ_CYCLE_COMPLETE_ACK

This bit is used by the interfaced module to acknowledge that the PLC has received a valid data packet. This should be turned on each time that TRIGGER_ SOURCE_ ACTIVE bit in the status word is set.

TRIGGER_SELECT

This bit of the function word tells the module whether the scanner trigger command is obtained from the PLC (the DEVICENET_READ_TRIGGER bit) or a discrete input into the module from a PE or Proximity etc.

Note: the barcode scanner needs to be configured to be trigger via the serial interface and not by its own discreet input.

DISCRETE_OUTPUT

This bit turns the discrete outputs on and off.

All other bits in the function word are reserved for later revisions

**Receive Data (Input to PLC/scanner, also known as Production Data):**

| Element | Description | Size (words) |
|---|---|---|
| 0 | Status Word ( see Status Word bit definition table below ) | 1 |
| 1-n | Received barcode characters (Ascii), length (n) set by parameter 8 | Value of parameter 8 |

## Status Word Bit Definition

The *Status Word* indicates the present status of the TCS module and the Serial Device.

| Bit | Description |
|---|---|
| 0 | TRIGGER_SOURCE_ACTIVE |
| 1 | READ_CYCLE_COMPLETE |
| 2 | GOOD_READ_INDICATION |
| 3 | READ_FAILURE |
| 4 – 7 | (Reserved) |
| 8 | DISCRETE_INPUT |
| 9 | (Reserved) |
| 10 | DISCRETE_OUTPUT_FAULT |
| 11 – 15 | (Reserved) |

## Bit Definitions

### TRIGGER_SOURCE_ACTIVE

Indicates that the trigger source is active.

### READ_CYCLE_COMPLETE

When a read has been completed, regardless whether it is a good or bad read, this bit will be set. This bit is reset when the READ_CYCLE_COMPLETE_ACK bit is set.

### GOOD_READ_INDICATION

This bit is set if a valid bar code is read.

### READ_FAILURE

This bit is set if a bad bar code is read. The bar code value will be FAILFAIL...

### DISCRETE_INPUT

This bit is the value of the discrete digital input.

### DISCRETE_OUTPUT_FAULT

This bit is set if the DISCRETE_OUTPUT value is different to the DISCRETE_OUTPUT bit.

All other bits in the status word are reserved for later revisions.

## DeviceNet Parameters

To change a parameter value, in DeviceNet select the appropriate parameter number and click on the Modify Parameter Button.

| **Parameter 1** | Factory Set |
|---|---|
| Description | Serial Number |
| Range | N/A |

| | |
|---|---|
| Default Value | N/A |
| Set By | Read Only |
| Function | To uniquely identify the product for quality control |

**Note**: This is a Read Only parameter, which is set at the factory. It may not be modified.

**Parameter 2**

| | |
|---|---|
| Description | DNet Mac ID |
| Range | 0 to 63 |
| Default Value | 63 |
| Set By | Read / Write |
| Function | To set the DeviceNet Mac ID / Address |

**Note**: When the TCS module is not in software select mode (DIP switches 7 and 8 are **not** both on), DIP switches 1 to 6 determine the DNet MAC ID.

When the TCS module is in software select mode (DIP switches 7 and 8 are on), this parameter can be set via the DeviceNet Manager.

**Parameter 3**

| | |
|---|---|
| Description | DNet Baudrate |
| Range | 125, 250, 500 kBaud (0,1,2) |
| Default Value | 125 kBaud |
| Set By | Read / Write |
| Function | To set the DeviceNet Network Baud Rate |

**Note**: This setting must be set to match the Baud rate of other devices on the network.

When the TCS module is not in software select mode (DIP switches 7 and 8 are **not** both on), DIP switches 7 and 8 determine the DNet Baudrate.

When the TCS module is in software select mode (DIP switches 7 and 8 are on), this parameter can be set via the DeviceNet Manager.

After changing the baud rate you will need to power the module down and up in order to effect the change.

**Parameter 4**

| | |
|---|---|
| Description | Comm1 Baudrate |
| Range | 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200 |
| Default Value | 9600 Baud |
| Set By | Read / Write |
| Function | To set the RS-232 / RS-485 serial Baud Rate |

**Note**: This parameter must be set to match the Baud rate of the Serial Device. After changing this parameter you will need to power the TCS module down and up in order to effect the change.

**Parameter 5**

| | |
|---|---|
| Description | Comm1 DataBits |
| Range | 7 or 8 |
| Default Value | 8 |
| Set By | Read / Write |
| Function | To set the RS-232 / RS-485 serial DataBits size |

**Note**: This parameter must be set to match the DataBits of the Serial Device. After changing this parameter you will need to power the TCS module down and up in order to effect the change.

**Parameter 6**

| | |
|---|---|
| Description | Comm1 Parity |
| Range | None, Odd or Even |
| Default Value | None |
| Set By | Read / Write |
| Function | To set the RS-232 / RS-485 serial Parity |

**Note**: This parameter must be set to match the Parity of the Serial Device. After changing this parameter you will need to power the TCS module down and up in order to effect the change.

**Parameter 7**

| | |
|---|---|
| Description | Comm1 StopBits |
| Range | 1 or 2 |
| Default Value | 1 |
| Set By | Read / Write |
| Function | To set the RS-232 / RS-485 serial Parity |

**Note**: This parameter must be set to match the DataBits of the Serial Device. After changing this parameter you will need to power the TCS module down and up in order to effect the change.

**Parameter 8**

| | |
|---|---|
| Description | Barcode Size |
| Range | 1 to 60 characters |
| Default Value | 10 |
| Set By | Read / Write |
| Function | To define the maximum number of characters to be read from a barcode. |

**Note**: This parameter should be set to match the largest number of characters to be read from the barcodes. This module uses a fragmented protocol so don't set this parameter larger than required, as it will create unnecessary bandwidth on the network. Two bytes + number of characters selected will be the TXD or production size. The value of this parameter will not take effect until the device is reset.

**Parameter 9**

| | |
|---|---|
| Description | Barcode trigger debounce |
| Range | 0 to 9.99 seconds (10 millisecond increments) |
| Default Value | 100 milliseconds |
| Set By | Read / Write |

| | |
|---|---|
| Function | To protect the start read command from momentary operation |

**Note**: This parameter will delay the start read trigger by the preset duration. It is intended to protect against accidental trigger only.

**Parameter 10**

| | |
|---|---|
| Description | Laser minimum on time |
| Range | 0 to 9.99 seconds (10 millisecond increments) |
| Default Value | 100 milliseconds |
| Set By | Read / Write |
| Function | To provide the laser with a minimum on period. |

**Note**: When the barcode scanner is triggered the laser is turned on for the duration of minimum on time, regardless whether or not a valid barcode is read during this period.

**Parameter 11**

| | |
|---|---|
| Description | Laser maximum on time |
| Range | 0 to 9.99 seconds (10 millisecond increments) |
| Default Value | 100 milliseconds |
| Set By | Read / Write |
| Function | To provide the laser with a maximum on period. |

**Note**: When the barcode scanner is triggered the laser is turned on for the duration of maximum on time. If no valid barcode is returned in this time then the laser is turned off and a failed response is returned.

**Parameter 12**

| | |
|---|---|
| Description | Trigger Mode |
| Range | Continuous (0) or One-Shot (1) |
| Default Value | Continuous (0) |
| Set By | Read / Write |
| Function | To set the laser to run continuously or on max / min time |

**Note**: The module can either trigger the scanner to run only for the duration of the scan or have the laser run continuously. For high speed operations it may be required to run the laser continuously to reduce spin up time.

**Parameter 13**

| | |
|---|---|
| Description | Trigger State |
| Range | Active On (0) or Active Low (1) |
| Default Value | Active On (0) |
| Set By | Read / Write |
| Function | to select whether the barcode scanner starts reading on the leading or trailing edge of the trigger sensor. |

**Note**: You may require to trigger the barcode scanner read as the barcode arrives or leaves the scanning point.

**Parameter 14**

| | |
|---|---|
| Description | Padding Character |
| Range | 0-1 |
| Default Value | Off (0) |
| Set By | Read / Write |
| Function | Fill in character for barcodes less than 30 characters. |

## Specifications

| TCS-DNSI-SICK | DeviceNet to Serial Interface Module (RS-232) |
|---|---|
| Supply Voltage | 24V DC DeviceNet™ Standard |
| Current | 50mA |
| Output | DeviceNet™ Standard |
| Digital Inputs | 10-30VDC optically isolated Currents:<br>2.5mA (min) @15V<br>12mA (max) @ 30V |

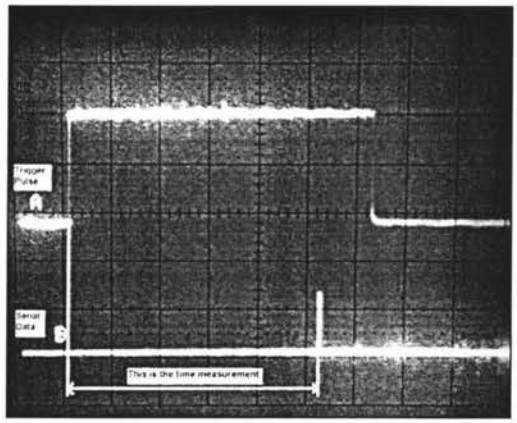
| Digital Outputs | 1A DC PNP ouputs |
|---|---|
| Electrical Isolation | Opto-couplers Isolation Voltage - 2.5KV rms min |
| Operating Temperature Range | -18°F to 158°F (-25°C to 70°C) |
| Enclosure | Anodised Aluminium |
| Relative Humidity | 5% to 95% |
| Connections DeviceNet | 5 pin push fit Pheonix Connector |
| Serial Connections | 9 pin D shell |
| Dimensions | 30mm(W) * 82mm(H) * 115mm(D) |
| Mounting | Din rail mounting in accordance with EN 50 022 |
| Serial Default Settings | Settings 9600, 8, 1, N |
| Serial Baud Rate | 115200, 57600, 38400, 19200, 14400, 9600, 4800, 2400, 1200 |
| Network Address | (00 to 63) |
| Baud Rate | 125, 250 or 500 Kb/s |
| User Configurable | MAC ID and Baud Rate |
| Software | DNSI-SICK |
| Product Code | 67 |
| Major Revision | 4 |
| Consumption Size (Tx Data) | 2 bytes |
| Production Data Size (Rx Data) | 2 + bar code size bytes |

# Appendix C Raw Data From Evaluation of Java Testing

## Sample Oscilloscope Screen for Measuring the Trigger Delay

The figure below is a photograph showing the measurement that was made on the oscilloscope screen. To get a stable image on the screen, a storage oscilloscope was used. Trace A is the trigger input and trace B shows the serial data going to the bar code scanner.

**Figure C.1 Sample oscilloscope readout for measuring the trigger delay**




## Raw Data Measured over DeviceNet

The tables below list the raw data that the time measurements were taken from. The first column is the index, this is referring to the packet number received by the packet capturing PC. The information in this column is discarded. The second column is the time index when the packet is received. The last column is the time calculated from the second column time, which is the time between the packets that initiate the action to the time the response message is sent. This is presented in tables 4.2 in chapter 4. The 'Event Description' is a description the author has added to the table to explain the packet data. The other columns contain the data and packet header information.

## Table C.1 The time delays of the C application

| Index | Time (ms) | DeviceNet I.D.# | Data | Event Description | Time measurement (ms) |
|---|---|---|---|---|---|
| 18 | 606567.146 | <07>2:05 [43D]2 | 11 0 | Bcode read trig | |
| 19 | 606570.86 | <07>1:15 [3C7]0 | | | |
| 20 | 606680.265 | <07>1:13 [347]8 | 0 7 0 33 39 35 31 39 | Barcode read | 113.119 |
| 21 | 606681.376 | <07>1:13 [347]8 | 41 39 30 39 35 38 37 37 | | |
| 22 | 606682.513 | <07>1:13 [347]8 | 42 20 30 20 20 20 20 20 | | |
| 23 | 606683.649 | <07>1:13 [347]8 | 43 20 20 20 20 20 20 20 | | |
| 24 | 606684.792 | <07>1:13 [347]8 | 44 20 20 20 20 20 20 20 | | |
| 25 | 606685.793 | <07>1:13 [347]6 | 85 20 20 20 20 20 | | |
| 26 | 606687.12 | <07>2:02 [43A]0 | | | |
| 27 | 606712.391 | <07>1:13 [347]8 | 0 7 0 33 39 35 31 39 | | |
| 28 | 606713.504 | <07>1:13 [347]8 | 41 39 30 39 35 38 37 37 | | |
| 29 | 606714.641 | <07>1:13 [347]8 | 42 20 30 20 20 20 20 20 | | |
| 30 | 606715.775 | <07>1:13 [347]8 | 43 20 20 20 20 20 20 20 | | |
| 31 | 606720.096 | <07>1:13 [347]8 | 44 20 20 20 20 20 20 20 | | |
| 32 | 606721.071 | <07>1:13 [347]6 | 85 20 20 20 20 20 | | |
| 33 | 606722.247 | <07>2:02 [43A]0 | | | |
| 34 | 606780.303 | <07>1:13 [347]8 | 0 6 0 33 39 35 31 39 | | |
| 35 | 606781.416 | <07>1:13 [347]8 | 41 39 30 39 35 38 37 37 | | |
| 36 | 606782.55 | <07>1:13 [347]8 | 42 20 30 20 20 20 20 20 | | |
| 37 | 606783.687 | <07>1:13 [347]8 | 43 20 20 20 20 20 20 20 | | |
| 38 | 606784.864 | <07>1:13 [347]8 | 44 20 20 20 20 20 20 20 | | |
| 39 | 606785.839 | <07>1:13 [347]6 | 85 20 20 20 20 20 | | |
| 40 | 606787.161 | <07>2:02 [43A]0 | | | |
| 41 | 606812.478 | <07>1:13 [347]8 | 0 6 0 33 39 35 31 39 | | |
| 42 | 606813.622 | <07>1:13 [347]8 | 41 39 30 39 35 38 37 37 | | |
| 43 | 606814.758 | <07>1:13 [347]8 | 42 20 30 20 20 20 20 20 | | |
| 44 | 606815.895 | <07>1:13 [347]8 | 43 20 20 20 20 20 20 20 | | |
| 45 | 606820.125 | <07>1:13 [347]8 | 44 20 20 20 20 20 20 20 | | |
| 46 | 606821.102 | <07>1:13 [347]6 | 85 20 20 20 20 20 | | |
| 47 | 606823.447 | <07>2:02 [43A]0 | | | |
| 48 | 607434.782 | <07>2:05 [43D]2 | 10 0 | | |
| 49 | 607439.783 | <07>1:15 [3C7]0 | | | |
| 50 | 608093.224 | <07>2:05 [43D]2 | 12 0 | PLC Ack rsp | |
| 51 | 608095.201 | <07>1:15 [3C7]0 | | | |
| 52 | 608100.73 | <07>1:13 [347]8 | 0 0 0 33 39 35 31 39 | Ack acked | 7.506 |
| 53 | 608101.841 | <07>1:13 [347]8 | 41 39 30 39 35 38 37 37 | | |
| 54 | 608102.978 | <07>1:13 [347]8 | 42 20 30 20 20 20 20 20 | | |
| 55 | 608104.113 | <07>1:13 [347]8 | 43 20 20 20 20 20 20 20 | | |
| 56 | 608105.297 | <07>1:13 [347]8 | 44 20 20 20 20 20 20 20 | | |
| 57 | 608106.272 | <07>1:13 [347]6 | 85 20 20 20 20 20 | | |
| 58 | 608107.465 | <07>2:02 [43A]0 | | | |
| 59 | 608133.496 | <07>1:13 [347]8 | 0 0 0 33 39 35 31 39 | | |
| 60 | 608134.608 | <07>1:13 [347]8 | 41 39 30 39 35 38 37 37 | | |
| 61 | 608135.744 | <07>1:13 [347]8 | 42 20 30 20 20 20 20 20 | | |
| 62 | 608136.911 | <07>1:13 [347]8 | 43 20 20 20 20 20 20 20 | | |
| 63 | 608140.657 | <07>1:13 [347]8 | 44 20 20 20 20 20 20 20 | | |
| 64 | 608141.632 | <07>1:13 [347]6 | 85 20 20 20 20 20 | | |
| 65 | 608142.695 | <07>2:02 [43A]0 | | | |
| 66 | 608880.261 | <07>2:05 [43D]2 | 10 0 | | |
| 67 | 608882.196 | <07>1:15 [3C7]0 | | | |
| 68 | 609542.718 | <07>2:05 [43D]2 | 11 0 | Bcode read trig | |
| 69 | 609544.672 | <07>1:15 [3C7]0 | | | |
| 70 | 609651.239 | <07>1:13 [347]8 | 0 1 0 33 39 35 31 39 | | |
| 71 | 609652.349 | <07>1:13 [347]8 | 41 39 30 39 35 38 37 37 | | |
| 72 | 609653.485 | <07>1:13 [347]8 | 42 20 30 20 20 20 20 20 | | |
| 73 | 609654.652 | <07>1:13 [347]8 | 43 20 20 20 20 20 20 20 | | |
| 74 | 609655.796 | <07>1:13 [347]8 | 44 20 20 20 20 20 20 20 | | |
| 75 | 609656.82 | <07>1:13 [347]6 | 85 20 20 20 20 20 | | |
| 76 | 609659.708 | <07>2:02 [43A]0 | | | |
| 77 | 609681.242 | <07>1:13 [347]8 | 0 7 0 33 39 35 31 39 | Barcode read | 138.524 |
| 78 | 609682.354 | <07>1:13 [347]8 | 41 39 30 39 35 38 37 37 | | |
| 79 | 609683.492 | <07>1:13 [347]8 | 42 20 30 20 20 20 20 20 | | |
| 80 | 609684.658 | <07>1:13 [347]8 | 43 20 20 20 20 20 20 20 | | |
| 81 | 609685.802 | <07>1:13 [347]8 | 44 20 20 20 20 20 20 20 | | |
| 82 | 609686.825 | <07>1:13 [347]6 | 85 20 20 20 20 20 | | |
| 83 | 609688.794 | <07>2:02 [43A]0 | | | |
| 84 | 609713.72 | <07>1:13 [347]8 | 0 7 0 33 39 35 31 39 | | |
| 85 | 609714.841 | <07>1:13 [347]8 | 41 39 30 39 35 38 37 37 | | |
| 86 | 609715.975 | <07>1:13 [347]8 | 42 20 30 20 20 20 20 20 | | |
| 87 | 609717.112 | <07>1:13 [347]8 | 43 20 20 20 20 20 20 20 | | |
| 88 | 609721.081 | <07>1:13 [347]8 | 44 20 20 20 20 20 20 20 | | |
| 89 | 609722.081 | <07>1:13 [347]6 | 85 20 20 20 20 20 | | |
| 90 | 609724.209 | <07>2:02 [43A]0 | | | |
| 91 | 609750.897 | <07>1:13 [347]8 | 0 6 0 33 39 35 31 39 | | |
| 92 | 609752.024 | <07>1:13 [347]8 | 41 39 30 39 35 38 37 37 | | |
| 93 | 609753.161 | <07>1:13 [347]8 | 42 20 30 20 20 20 20 20 | | |
| 94 | 609754.295 | <07>1:13 [347]8 | 43 20 20 20 20 20 20 20 | | |
| 95 | 609755.489 | <07>1:13 [347]8 | 44 20 20 20 20 20 20 20 | | |

```
96   609756.462<07>1:13 [347]6  85 20 20 20 20 20
97   609758.8  <07>2:02 [43A]0
98   610300.438<07>2:05 [43D]2  10 0
99   610302.375<07>1:15 [3C7]0
100  610994.272<07>2:05 [43D]2  12 0                                  PLC Ack rsp
101  610996.225<07>1:15 [3C7]0
102  611001.674<07>1:13 [347]8  0  0  0  33 39 35 31 39   Ack acked        7.402
103  611002.786<07>1:13 [347]8  41 39 30 39 35 38 37 37
104  611003.924<07>1:13 [347]8  42 20 30 20 20 20 20 20
105  611005.098<07>1:13 [347]8  43 20 20 20 20 20 20 20
106  611006.242<07>1:13 [347]8  44 20 20 20 20 20 20 20
107  611007.258<07>1:13 [347]6  85 20 20 20 20 20
108  611008.569<07>2:02 [43A]0
109  611033.744<07>1:13 [347]8  0  0  0  33 39 35 31 39
110  611034.872<07>1:13 [347]8  41 39 30 39 35 38 37 37
111  611036.008<07>1:13 [347]8  42 20 30 20 20 20 20 20
112  611037.146<07>1:13 [347]8  43 20 20 20 20 20 20 20
113  611041.505<07>1:13 [347]8  44 20 20 20 20 20 20 20
114  611042.522<07>1:13 [347]6  85 20 20 20 20 20
115  611043.696<07>2:02 [43A]0
116  611960.681<07>2:05 [43D]2  10 0
117  611962.64 <07>1:15 [3C7]0
118  612646.834<07>2:05 [43D]2  11 0                                  Bcode read trig
119  612651.85 <07>1:15 [3C7]0
120  612762.248<07>1:13 [347]8  0  1  0  33 39 35 31 39
121  612763.36 <07>1:13 [347]8  41 39 30 39 35 38 37 37
122  612764.496<07>1:13 [347]8  42 20 30 20 20 20 20 20
123  612765.632<07>1:13 [347]8  43 20 20 20 20 20 20 20
124  612766.776<07>1:13 [347]8  44 20 20 20 20 20 20 20
125  612767.776<07>1:13 [347]6  85 20 20 20 20 20
126  612769.106<07>2:02 [43A]0
127  612792.262<07>1:13 [347]8  0  7  0  33 39 35 31 39   Barcode read    145.428
128  612793.374<07>1:13 [347]8  41 39 30 39 35 38 37 37
129  612794.509<07>1:13 [347]8  42 20 30 20 20 20 20 20
130  612795.646<07>1:13 [347]8  43 20 20 20 20 20 20 20
131  612796.837<07>1:13 [347]8  44 20 20 20 20 20 20 20
132  612797.813<07>1:13 [347]6  85 20 20 20 20 20
133  612798.974<07>2:02 [43A]0
134  612825.141<07>1:13 [347]8  0  7  0  33 39 35 31 39
135  612826.252<07>1:13 [347]8  41 39 30 39 35 38 37 37
136  612827.436<07>1:13 [347]8  42 20 30 20 20 20 20 20
137  612831.645<07>1:13 [347]8  43 20 20 20 20 20 20 20
138  612832.797<07>1:13 [347]8  44 20 20 20 20 20 20 20
139  612833.774<07>1:13 [347]6  85 20 20 20 20 20
140  612835.446<07>2:02 [43A]0
141  612862.293<07>1:13 [347]8  0  6  0  33 39 35 31 39
142  612863.405<07>1:13 [347]8  41 39 30 39 35 38 37 37
143  612864.541<07>1:13 [347]8  42 20 30 20 20 20 20 20
144  612865.676<07>1:13 [347]8  43 20 20 20 20 20 20 20
145  612866.844<07>1:13 [347]8  44 20 20 20 20 20 20 20
146  612867.819<07>1:13 [347]6  85 20 20 20 20 20
147  612869.901<07>2:02 [43A]0
148  612895.196<07>1:13 [347]8  0  6  0  33 39 35 31 39
149  612896.316<07>1:13 [347]8  41 39 30 39 35 38 37 37
150  612897.453<07>1:13 [347]8  42 20 30 20 20 20 20 20
151  612901.651<07>1:13 [347]8  43 20 20 20 20 20 20 20
152  612902.812<07>1:13 [347]8  44 20 20 20 20 20 20 20
153  612903.787<07>1:13 [347]6  85 20 20 20 20 20
154  612905.835<07>2:02 [43A]0
155  613522.428<07>2:05 [43D]2  10 0
156  613524.364<07>1:15 [3C7]0
157  614391.175<07>2:05 [43D]2  12 0                                  PLC Ack rsp
158  614393.415<07>1:15 [3C7]0
159  614402.785<07>1:13 [347]8  0  0  0  33 39 35 31 39   Ack acked        11.61
160  614403.898<07>1:13 [347]8  41 39 30 39 35 38 37 37
161  614405.035<07>1:13 [347]8  42 20 30 20 20 20 20 20
162  614406.202<07>1:13 [347]8  43 20 20 20 20 20 20 20
163  614407.345<07>1:13 [347]8  44 20 20 20 20 20 20 20
164  614408.369<07>1:13 [347]6  85 20 20 20 20 20
165  614409.529<07>2:02 [43A]0
166  614435.384<07>1:13 [347]8  0  0  0  33 39 35 31 39
167  614436.496<07>1:13 [347]8  41 39 30 39 35 38 37 37
168  614437.632<07>1:13 [347]8  42 20 30 20 20 20 20 20
169  614438.768<07>1:13 [347]8  43 20 20 20 20 20 20 20
170  614442.617<07>1:13 [347]8  44 20 20 20 20 20 20 20
171  614443.625<07>1:13 [347]6  85 20 20 20 20 20
172  614444.649<07>2:02 [43A]0
173  615470.393<07>2:05 [43D]2  10 0
174  615473.755<07>1:15 [3C7]0
175  616132.572<07>2:05 [43D]2  11 0                                  Bcode read trig
176  616134.524<07>1:15 [3C7]0
177  616243.386<07>1:13 [347]8  0  1  0  33 39 35 31 39
178  616244.5  <07>1:13 [347]8  41 39 30 39 35 38 37 37
179  616245.634<07>1:13 [347]8  42 20 30 20 20 20 20 20
180  616246.769<07>1:13 [347]8  43 20 20 20 20 20 20 20
```

```
181  616247.945<07>1:13 [347]8  44 20 20 20 20 20 20 20
182  616248.922<07>1:13 [347]6  85 20 20 20 20 20
183  616250.274<07>2:02 [43A]0
184  616273.4  <07>1:13 [347]8  0  7  0  33 39 35 31 39   Barcode read      140.828
185  616274.511<07>1:13 [347]8  41 39 30 39 35 38 37 37
186  616275.647<07>1:13 [347]8  42 20 30 20 20 20 20 20
187  616276.784<07>1:13 [347]8  43 20 20 20 20 20 20 20
188  616277.951<07>1:13 [347]8  44 20 20 20 20 20 20 20
189  616278.926<07>1:13 [347]6  85 20 20 20 20 20
190  616280.095<07>2:02 [43A]0
191  616305.838<07>1:13 [347]8  0  7  0  33 39 35 31 39
192  616306.95 <07>1:13 [347]8  41 39 30 39 35 38 37 37
193  616308.087<07>1:13 [347]8  42 20 30 20 20 20 20 20
194  616309.261<07>1:13 [347]8  43 20 20 20 20 20 20 20
195  616313.23 <07>1:13 [347]8  44 20 20 20 20 20 20 20
196  616314.207<07>1:13 [347]6  85 20 20 20 20 20
197  616315.367<07>2:02 [43A]0
198  616343.431<07>1:13 [347]8  0  6  0  33 39 35 31 39
199  616344.541<07>1:13 [347]8  41 39 30 39 35 38 37 37
200  616345.676<07>1:13 [347]8  42 20 30 20 20 20 20 20
201  616346.847<07>1:13 [347]8  43 20 20 20 20 20 20 20
202  616347.989<07>1:13 [347]8  44 20 20 20 20 20 20 20
203  616349.013<07>1:13 [347]6  85 20 20 20 20 20
204  616350.99 <07>2:02 [43A]0
205  616375.9  <07>1:13 [347]8  0  6  0  33 39 35 31 39
206  616377.029<07>1:13 [347]8  41 39 30 39 35 38 37 37
207  616378.164<07>1:13 [347]8  42 20 30 20 20 20 20 20
208  616379.299<07>1:13 [347]8  43 20 20 20 20 20 20 20
209  616383.252<07>1:13 [347]8  44 20 20 20 20 20 20 20
210  616384.276<07>1:13 [347]6  85 20 20 20 20 20
211  616386.878<07>2:02 [43A]0
212  616892.731<07>2:05 [43D]2  10 0
213  616894.668<07>1:15 [3C7]0
214  617611.331<07>2:05 [43D]2  12 0                      PLC Ack rsp
215  617614.467<07>1:15 [3C7]0
216  617623.837<07>1:13 [347]8  0  0  0  33 39 35 31 39   Ack acked         12.506
217  617624.951<07>1:13 [347]8  41 39 30 39 35 38 37 37
218  617626.086<07>1:13 [347]8  42 20 30 20 20 20 20 20
219  617627.221<07>1:13 [347]8  43 20 20 20 20 20 20 20
220  617628.397<07>1:13 [347]8  44 20 20 20 20 20 20 20
221  617629.372<07>1:13 [347]6  85 20 20 20 20 20
222  617632.549<07>2:02 [43A]0
223  618482.548<07>2:05 [43D]2  10 0
224  618484.742<07>1:15 [3C7]0
225  619171.253<07>2:05 [43D]2  11 0                      Bcode read trig
226  619174.983<07>1:15 [3C7]0
227  619284.379<07>1:13 [347]8  0  1  0  33 39 35 31 39
228  619285.525<07>1:13 [347]8  41 39 30 39 35 38 37 37
229  619286.66 <07>1:13 [347]8  42 20 30 20 20 20 20 20
230  619287.796<07>1:13 [347]8  43 20 20 20 20 20 20 20
231  619288.941<07>1:13 [347]8  44 20 20 20 20 20 20 20
232  619289.939<07>1:13 [347]6  85 20 20 20 20 20
233  619291.755<07>2:02 [43A]0
234  619314.396<07>1:13 [347]8  0  7  0  33 39 35 31 39   Barcode read      143.143
235  619315.53 <07>1:13 [347]8  41 39 30 39 35 38 37 37
236  619316.665<07>1:13 [347]8  42 20 30 20 20 20 20 20
237  619317.802<07>1:13 [347]8  43 20 20 20 20 20 20 20
238  619318.947<07>1:13 [347]8  44 20 20 20 20 20 20 20
239  619319.945<07>1:13 [347]6  85 20 20 20 20 20
240  619321.394<07>2:02 [43A]0
241  619347.201<07>1:13 [347]8  0  7  0  33 39 35 31 39
242  619348.313<07>1:13 [347]8  41 39 30 39 35 38 37 37
243  619349.449<07>1:13 [347]8  42 20 30 20 20 20 20 20
244  619353.681<07>1:13 [347]8  43 20 20 20 20 20 20 20
245  619354.865<07>1:13 [347]8  44 20 20 20 20 20 20 20
246  619355.843<07>1:13 [347]6  85 20 20 20 20 20
247  619357.882<07>2:02 [43A]0
248  619384.424<07>1:13 [347]8  0  6  0  33 39 35 31 39
249  619385.544<07>1:13 [347]8  41 39 30 39 35 38 37 37
250  619386.68 <07>1:13 [347]8  42 20 30 20 20 20 20 20
251  619387.817<07>1:13 [347]8  43 20 20 20 20 20 20 20
252  619388.96 <07>1:13 [347]8  44 20 20 20 20 20 20 20
253  619389.96 <07>1:13 [347]6  85 20 20 20 20 20
254  619392.217<07>2:02 [43A]0
255  619417.263<07>1:13 [347]8  0  6  0  33 39 35 31 39
256  619418.375<07>1:13 [347]8  41 39 30 39 35 38 37 37
257  619419.513<07>1:13 [347]8  42 20 30 20 20 20 20 20
258  619423.712<07>1:13 [347]8  43 20 20 20 20 20 20 20
259  619424.872<07>1:13 [347]8  44 20 20 20 20 20 20 20
260  619425.849<07>1:13 [347]6  85 20 20 20 20 20
261  619427.64 <07>2:02 [43A]0
262  620022.565<07>2:05 [43D]2  10 0
263  620025.244<07>1:15 [3C7]0
264  621133.888<07>2:05 [43D]2  12 0                      PLC Ack rsp
265  621135.839<07>1:15 [3C7]0
```

```
266  621144.986<07>1:13 [347]8  0  0  0  33 39 35 31 39   Ack acked        11.098
267  621146.098<07>1:13 [347]8  41 39 30 39 35 38 37 37
268  621147.233<07>1:13 [347]8  42 20 30 20 20 20 20 20
269  621148.401<07>1:13 [347]8  43 20 20 20 20 20 20 20
270  621149.544<07>1:13 [347]8  44 20 20 20 20 20 20 20
271  621150.568<07>1:13 [347]6  85 20 20 20 20 20
272  621151.865<07>2:02 [43A]0
273  621177.624<07>1:13 [347]8  0  0  0  33 39 35 31 39
274  621178.736<07>1:13 [347]8  41 39 30 39 35 38 37 37
275  621179.872<07>1:13 [347]8  42 20 30 20 20 20 20 20
276  621181.039<07>1:13 [347]8  43 20 20 20 20 20 20 20
277  621184.823<07>1:13 [347]8  44 20 20 20 20 20 20 20
278  621185.83 <07>1:13 [347]6  85 20 20 20 20 20
279  621187.463<07>2:02 [43A]0
280  622090.237<07>2:05 [43D]2  10 0
281  622095.277<07>1:15 [3C7]0
282  622755.039<07>2:05 [43D]2  11 0                      Bcode read trig
283  622756.991<07>1:15 [3C7]0
284  622865.549<07>1:13 [347]8  0  1  0  33 39 35 31 39
285  622866.66 <07>1:13 [347]8  41 39 30 39 35 38 37 37
286  622867.797<07>1:13 [347]8  42 20 30 20 20 20 20 20
287  622868.934<07>1:13 [347]8  43 20 20 20 20 20 20 20
288  622870.076<07>1:13 [347]8  44 20 20 20 20 20 20 20
289  622871.077<07>1:13 [347]6  85 20 20 20 20 20
290  622872.429<07>2:02 [43A]0
291  622895.563<07>1:13 [347]8  0  7  0  33 39 35 31 39   Barcode read    140.524
292  622896.675<07>1:13 [347]8  41 39 30 39 35 38 37 37
293  622897.81 <07>1:13 [347]8  42 20 30 20 20 20 20 20
294  622898.946<07>1:13 [347]8  43 20 20 20 20 20 20 20
295  622900.138<07>1:13 [347]8  44 20 20 20 20 20 20 20
296  622901.115<07>1:13 [347]6  85 20 20 20 20 20
297  622903.851<07>2:02 [43A]0
298  622927.986<07>1:13 [347]8  0  7  0  33 39 35 31 39
299  622929.096<07>1:13 [347]8  41 39 30 39 35 38 37 37
300  622930.234<07>1:13 [347]8  42 20 30 20 20 20 20 20
301  622931.369<07>1:13 [347]8  43 20 20 20 20 20 20 20
302  622935.394<07>1:13 [347]8  44 20 20 20 20 20 20 20
303  622936.37 <07>1:13 [347]6  85 20 20 20 20 20
304  622938.522<07>2:02 [43A]0
305  622965.594<07>1:13 [347]8  0  6  0  33 39 35 31 39
306  622966.705<07>1:13 [347]8  41 39 30 39 35 38 37 37
307  622967.844<07>1:13 [347]8  42 20 30 20 20 20 20 20
308  622968.978<07>1:13 [347]8  43 20 20 20 20 20 20 20
309  622970.145<07>1:13 [347]8  44 20 20 20 20 20 20 20
310  622971.122<07>1:13 [347]6  85 20 20 20 20 20
311  622972.594<07>2:02 [43A]0
312  622998.04 <07>1:13 [347]8  0  6  0  33 39 35 31 39
313  622999.152<07>1:13 [347]8  41 39 30 39 35 38 37 37
314  623000.287<07>1:13 [347]8  42 20 30 20 20 20 20 20
315  623001.455<07>1:13 [347]8  43 20 20 20 20 20 20 20
316  623005.417<07>1:13 [347]8  44 20 20 20 20 20 20 20
317  623006.392<07>1:13 [347]6  85 20 20 20 20 20
318  623009.097<07>2:02 [43A]0
319  623537.373<07>2:05 [43D]2  10 0
320  623539.332<07>1:15 [3C7]0
321  624261.599<07>2:05 [43D]2  12 0                      PLC Ack rsp
322  624266.646<07>1:15 [3C7]0
323  624276.01 <07>1:13 [347]8  0  0  0  33 39 35 31 39   Ack acked        14.411
324  624277.121<07>1:13 [347]8  41 39 30 39 35 38 37 37
325  624278.256<07>1:13 [347]8  42 20 30 20 20 20 20 20
326  624279.391<07>1:13 [347]8  43 20 20 20 20 20 20 20
327  624280.584<07>1:13 [347]8  44 20 20 20 20 20 20 20
328  624281.56 <07>1:13 [347]6  85 20 20 20 20 20
329  624283.76 <07>2:02 [43A]0
330  624308.055<07>1:13 [347]8  0  0  0  33 39 35 31 39
331  624309.166<07>1:13 [347]8  41 39 30 39 35 38 37 37
332  624310.302<07>1:13 [347]8  42 20 30 20 20 20 20 20
333  624311.438<07>1:13 [347]8  43 20 20 20 20 20 20 20
334  624315.848<07>1:13 [347]8  44 20 20 20 20 20 20 20
335  624316.822<07>1:13 [347]6  85 20 20 20 20 20
336  624318.223<07>2:02 [43A]0
337  625112.432<07>2:05 [43D]2  10 0
338  625116.906<07>1:15 [3C7]0
339  625890.471<07>2:05 [43D]2  11 0                      Bcode read trig
340  625895.488<07>1:15 [3C7]0
341  626006.582<07>1:13 [347]8  0  1  0  33 39 35 31 39
342  626007.726<07>1:13 [347]8  41 39 30 39 35 38 37 37
343  626008.862<07>1:13 [347]8  42 20 30 20 20 20 20 20
344  626009.998<07>1:13 [347]8  43 20 20 20 20 20 20 20
345  626011.142<07>1:13 [347]8  44 20 20 20 20 20 20 20
346  626012.133<07>1:13 [347]6  85 20 20 20 20 20
347  626013.526<07>2:02 [43A]0
348  626036.587<07>1:13 [347]8  0  7  0  33 39 35 31 39   Barcode read    146.116
349  626037.732<07>1:13 [347]8  41 39 30 39 35 38 37 37
350  626038.867<07>1:13 [347]8  42 20 30 20 20 20 20 20
```

```
351  626040.003<07>1:13 [347]8  43 20 20 20 20 20 20 20
352  626041.147<07>1:13 [347]8  44 20 20 20 20 20 20 20
353  626042.242<07>1:13 [347]6  85 20 20 20 20 20
354  626043.754<07>2:02 [43A]0
355  626069.417<07>1:13 [347]8  0  7  0  33 39 35 31 39
356  626070.561<07>1:13 [347]8  41 39 30 39 35 38 37 37
357  626071.698<07>1:13 [347]8  42 20 30 20 20 20 20 20
358  626075.897<07>1:13 [347]8  43 20 20 20 20 20 20 20
359  626077.058<07>1:13 [347]8  44 20 20 20 20 20 20 20
360  626078.033<07>1:13 [347]6  85 20 20 20 20 20
361  626079.065<07>2:02 [43A]0
362  626106.617<07>1:13 [347]8  0  6  0  33 39 35 31 39
363  626107.745<07>1:13 [347]8  41 39 30 39 35 38 37 37
364  626108.88 <07>1:13 [347]8  42 20 30 20 20 20 20 20
365  626110.018<07>1:13 [347]8  43 20 20 20 20 20 20 20
366  626111.208<07>1:13 [347]8  44 20 20 20 20 20 20 20
367  626112.256<07>1:13 [347]6  85 20 20 20 20 20
368  626113.545<07>2:02 [43A]0
369  626139.481<07>1:13 [347]8  0  6  0  33 39 35 31 39
370  626140.592<07>1:13 [347]8  41 39 30 39 35 38 37 37
371  626141.729<07>1:13 [347]8  42 20 30 20 20 20 20 20
372  626145.927<07>1:13 [347]8  43 20 20 20 20 20 20 20
373  626147.071<07>1:13 [347]8  44 20 20 20 20 20 20 20
374  626148.047<07>1:13 [347]6  85 20 20 20 20 20
375  626149.671<07>2:02 [43A]0
376  626677.186<07>2:05 [43D]2  10 0
377  626679.125<07>1:15 [3C7]0
378  627373.685<07>2:05 [43D]2  12 0                          PLC Ack rsp
379  627377.662<07>1:15 [3C7]0
380  627387.025<07>1:13 [347]8  0  0  0  33 39 35 31 39   Ack acked        13.34
381  627388.169<07>1:13 [347]8  41 39 30 39 35 38 37 37
382  627389.304<07>1:13 [347]8  42 20 30 20 20 20 20 20
383  627390.441<07>1:13 [347]8  43 20 20 20 20 20 20 20
384  627391.585<07>1:13 [347]8  44 20 20 20 20 20 20 20
385  627392.585<07>1:13 [347]6  85 20 20 20 20 20
386  627393.92 <07>2:02 [43A]0
387  627419.472<07>1:13 [347]8  0  0  0  33 39 35 31 39
388  627420.598<07>1:13 [347]8  41 39 30 39 35 38 37 37
389  627421.735<07>1:13 [347]8  42 20 30 20 20 20 20 20
390  627422.871<07>1:13 [347]8  43 20 20 20 20 20 20 20
391  627426.866<07>1:13 [347]8  44 20 20 20 20 20 20 20
392  627427.84 <07>1:13 [347]6  85 20 20 20 20 20
393  627430.167<07>2:02 [43A]0
394  628187.291<07>2:05 [43D]2  10 0
395  628189.242<07>1:15 [3C7]0
396  629752.666<07>2:05 [43D]2  11 0                          Bcode read trig
397  629757.681<07>1:15 [3C7]0
398  629867.84 <07>1:13 [347]8  0  1  0  33 39 35 31 39
399  629868.952<07>1:13 [347]8  41 39 30 39 35 38 37 37
400  629870.087<07>1:13 [347]8  42 20 30 20 20 20 20 20
401  629871.359<07>1:13 [347]8  43 20 20 20 20 20 20 20
402  629872.503<07>1:13 [347]8  44 20 20 20 20 20 20 20
403  629873.527<07>1:13 [347]6  85 20 20 20 20 20
404  629875.647<07>2:02 [43A]0
405  629897.852<07>1:13 [347]8  0  7  0  33 39 35 31 39   Barcode read    145.186
406  629898.964<07>1:13 [347]8  41 39 30 39 35 38 37 37
407  629900.1  <07>1:13 [347]8  42 20 30 20 20 20 20 20
408  629901.267<07>1:13 [347]8  43 20 20 20 20 20 20 20
409  629902.414<07>1:13 [347]8  44 20 20 20 20 20 20 20
410  629903.428<07>1:13 [347]6  85 20 20 20 20 20
411  629906.317<07>2:02 [43A]0
412  629930.418<07>1:13 [347]8  0  7  0  33 39 35 31 39
413  629931.546<07>1:13 [347]8  41 39 30 39 35 38 37 37
414  629932.683<07>1:13 [347]8  42 20 30 20 20 20 20 20
415  629933.819<07>1:13 [347]8  43 20 20 20 20 20 20 20
416  629937.683<07>1:13 [347]8  44 20 20 20 20 20 20 20
417  629938.692<07>1:13 [347]6  85 20 20 20 20 20
418  629940.099<07>2:02 [43A]0
419  629967.882<07>1:13 [347]8  0  6  0  33 39 35 31 39
420  629969.034<07>1:13 [347]8  41 39 30 39 35 38 37 37
421  629970.172<07>1:13 [347]8  42 20 30 20 20 20 20 20
422  629971.306<07>1:13 [347]8  43 20 20 20 20 20 20 20
423  629972.498<07>1:13 [347]8  44 20 20 20 20 20 20 20
424  629973.474<07>1:13 [347]6  85 20 20 20 20 20
425  629974.923<07>2:02 [43A]0
426  630000.48 <07>1:13 [347]8  0  6  0  33 39 35 31 39
427  630001.593<07>1:13 [347]8  41 39 30 39 35 38 37 37
428  630002.728<07>1:13 [347]8  42 20 30 20 20 20 20 20
429  630003.912<07>1:13 [347]8  43 20 20 20 20 20 20 20
430  630007.705<07>1:13 [347]8  44 20 20 20 20 20 20 20
431  630008.705<07>1:13 [347]6  85 20 20 20 20 20
432  630011.441<07>2:02 [43A]0
433  630746.013<07>2:05 [43D]2  10 0
434  630748.743<07>1:15 [3C7]0
435  631409.185<07>2:05 [43D]2  12 0                          PLC Ack rsp
```

```
436  631411.145<07>1:15 [3C7] 0
437  631418.347<07>1:13 [347] 8  0  0  0  33 39 35 31 39   Ack acked        9.162
438  631419.491<07>1:13 [347] 8  41 39 30 39 35 38 37 37
439  631420.628<07>1:13 [347] 8  42 20 30 20 20 20 20 20
440  631421.762<07>1:13 [347] 8  43 20 20 20 20 20 20 20
441  631422.909<07>1:13 [347] 8  44 20 20 20 20 20 20 20
442  631423.899<07>1:13 [347] 6  85 20 20 20 20 20
443  631425.355<07>2:02 [43A] 0
444  631450.602<07>1:13 [347] 8  0  0  0  33 39 35 31 39
445  631451.714<07>1:13 [347] 8  41 39 30 39 35 38 37 37
446  631452.849<07>1:13 [347] 8  42 20 30 20 20 20 20 20
447  631453.985<07>1:13 [347] 8  43 20 20 20 20 20 20 20
448  631458.177<07>1:13 [347] 8  44 20 20 20 20 20 20 20
449  631459.161<07>1:13 [347] 6  85 20 20 20 20 20
450  631460.995<07>2:02 [43A] 0
451  632252.828<07>2:05 [43D] 2  10 0
452  632258.245<07>1:15 [3C7] 0
453  632974.692<07>2:05 [43D] 2  11 0                      Bcode read trig
454  632979.493<07>1:15 [3C7] 0
455  633088.891<07>1:13 [347] 8  0  1  0  33 39 35 31 39
456  633090.005<07>1:13 [347] 8  41 39 30 39 35 38 37 37
457  633091.139<07>1:13 [347] 8  42 20 30 20 20 20 20 20
458  633092.275<07>1:13 [347] 8  43 20 20 20 20 20 20 20
459  633093.452<07>1:13 [347] 8  44 20 20 20 20 20 20 20
460  633094.427<07>1:13 [347] 6  85 20 20 20 20 20
461  633095.65 <07>2:02 [43A] 0
462  633118.907<07>1:13 [347] 8  0  5  0  33 39 35 31 39
463  633120.017<07>1:13 [347] 8  41 39 30 39 35 38 37 37
464  633121.153<07>1:13 [347] 8  42 20 30 20 20 20 20 20
465  633122.289<07>1:13 [347] 8  43 20 20 20 20 20 20 20
466  633123.456<07>1:13 [347] 8  44 20 20 20 20 20 20 20
467  633124.434<07>1:13 [347] 6  85 20 20 20 20 20
468  633126.041<07>2:02 [43A] 0
469  633128.987<07>1:13 [347] 8  0  7  0  33 39 35 31 39   Barcode read    154.295
470  633130.096<07>1:13 [347] 8  41 39 30 39 35 38 37 37
471  633131.235<07>1:13 [347] 8  42 20 30 20 20 20 20 20
472  633132.369<07>1:13 [347] 8  43 20 20 20 20 20 20 20
473  633133.512<07>1:13 [347] 8  44 20 20 20 20 20 20 20
474  633134.504<07>1:13 [347] 6  85 20 20 20 20 20
475  633135.52 <07>2:02 [43A] 0
476  633160.928<07>1:13 [347] 8  0  7  0  33 39 35 31 39
477  633162.039<07>1:13 [347] 8  41 39 30 39 35 38 37 37
478  633163.174<07>1:13 [347] 8  42 20 30 20 20 20 20 20
479  633164.313<07>1:13 [347] 8  43 20 20 20 20 20 20 20
480  633168.736<07>1:13 [347] 8  44 20 20 20 20 20 20 20
481  633169.711<07>1:13 [347] 6  85 20 20 20 20 20
482  633170.903<07>2:02 [43A] 0
483  633188.932<07>1:13 [347] 8  0  6  0  33 39 35 31 39
484  633190.069<07>1:13 [347] 8  41 39 30 39 35 38 37 37
485  633191.18 <07>1:13 [347] 8  42 20 30 20 20 20 20 20
486  633192.355<07>1:13 [347] 8  43 20 20 20 20 20 20 20
487  633193.499<07>1:13 [347] 8  44 20 20 20 20 20 20 20
488  633194.517<07>1:13 [347] 6  85 20 20 20 20 20
489  633197.404<07>2:02 [43A] 0
490  633220.979<07>1:13 [347] 8  0  6  0  33 39 35 31 39
491  633222.13 <07>1:13 [347] 8  41 39 30 39 35 38 37 37
492  633223.265<07>1:13 [347] 8  42 20 30 20 20 20 20 20
493  633224.401<07>1:13 [347] 8  43 20 20 20 20 20 20 20
494  633228.755<07>1:13 [347] 8  44 20 20 20 20 20 20 20
495  633229.78 <07>1:13 [347] 6  85 20 20 20 20 20
496  633230.795<07>2:02 [43A] 0
497  633910.947<07>2:05 [43D] 2  10 0
498  633912.882<07>1:15 [3C7] 0
499  634663.626<07>2:05 [43D] 2  12 0                      PLC Ack rsp
500  634668.644<07>1:15 [3C7] 0
501  634679.414<07>1:13 [347] 8  0  0  0  33 39 35 31 39   Ack acked       15.788
502  634680.574<07>1:13 [347] 8  41 39 30 39 35 38 37 37
503  634681.711<07>1:13 [347] 8  42 20 30 20 20 20 20 20
504  634682.846<07>1:13 [347] 8  43 20 20 20 20 20 20 20
505  634684.04 <07>1:13 [347] 8  44 20 20 20 20 20 20 20
506  634685.014<07>1:13 [347] 6  85 20 20 20 20 20
507  634686.301<07>2:02 [43A] 0
508  634712.134<07>1:13 [347] 8  0  0  0  33 39 35 31 39
509  634713.244<07>1:13 [347] 8  41 39 30 39 35 38 37 37
510  634714.38 <07>1:13 [347] 8  42 20 30 20 20 20 20 20
511  634715.548<07>1:13 [347] 8  43 20 20 20 20 20 20 20
512  634719.244<07>1:13 [347] 8  44 20 20 20 20 20 20 20
513  634720.245<07>1:13 [347] 6  85 20 20 20 20 20
514  634721.957<07>2:02 [43A] 0
515  635387.306<07>2:05 [43D] 2  10 0
516  635390.266<07>1:15 [3C7] 0
517  636798.156<07>2:05 [43D] 2  11 0                      Bcode read trig
518  636800.74 <07>1:15 [3C7] 0
519  636910.146<07>1:13 [347] 8  0  1  0  33 39 35 31 39
520  636911.266<07>1:13 [347] 8  41 39 30 39 35 38 37 37
```

```
521  636912.402<07>1:13 [347]8  42 20 30 20 20 20 20 20
522  636913.538<07>1:13 [347]8  43 20 20 20 20 20 20 20
523  636914.682<07>1:13 [347]8  44 20 20 20 20 20 20 20
524  636915.681<07>1:13 [347]6  85 20 20 20 20 20
525  636917.002<07>2:02 [43A]0
526  636940.151<07>1:13 [347]8  0  7  0  33 39 35 31 39   Barcode read     141.995
527  636941.271<07>1:13 [347]8  41 39 30 39 35 38 37 37
528  636942.408<07>1:13 [347]8  42 20 30 20 20 20 20 20
529  636943.545<07>1:13 [347]8  43 20 20 20 20 20 20 20
530  636944.687<07>1:13 [347]8  44 20 20 20 20 20 20 20
531  636945.687<07>1:13 [347]6  85 20 20 20 20 20
532  636947.679<07>2:02 [43A]0
533  636972.886<07>1:13 [347]8  0  7  0  33 39 35 31 39
534  636974.006<07>1:13 [347]8  41 39 30 39 35 38 37 37
535  636975.142<07>1:13 [347]8  42 20 30 20 20 20 20 20
536  636976.279<07>1:13 [347]8  43 20 20 20 20 20 20 20
537  636979.982<07>1:13 [347]8  44 20 20 20 20 20 20 20
538  636980.96 <07>1:13 [347]6  85 20 20 20 20 20
539  636983.134<07>2:02 [43A]0
540  637010.181<07>1:13 [347]8  0  6  0  33 39 35 31 39
541  637011.294<07>1:13 [347]8  41 39 30 39 35 38 37 37
542  637012.429<07>1:13 [347]8  42 20 30 20 20 20 20 20
543  637013.565<07>1:13 [347]8  43 20 20 20 20 20 20 20
544  637014.757<07>1:13 [347]8  44 20 20 20 20 20 20 20
545  637015.734<07>1:13 [347]6  85 20 20 20 20 20
546  637016.926<07>2:02 [43A]0
547  637042.948<07>1:13 [347]8  0  6  0  33 39 35 31 39
548  637044.06 <07>1:13 [347]8  41 39 30 39 35 38 37 37
549  637045.197<07>1:13 [347]8  42 20 30 20 20 20 20 20
550  637049.437<07>1:13 [347]8  43 20 20 20 20 20 20 20
551  637050.62 <07>1:13 [347]8  44 20 20 20 20 20 20 20
552  637051.598<07>1:13 [347]6  85 20 20 20 20 20
553  637053.205<07>2:02 [43A]0
554  637704.158<07>2:05 [43D]2  10 0
555  637709.158<07>1:15 [3C7]0
556  638365.769<07>2:05 [43D]2  12 0                     PLC Ack rsp
557  638370.794<07>1:15 [3C7]0
558  638380.62 <07>1:13 [347]8  0  0  0  33 39 35 31 39   Ack acked        14.851
559  638381.731<07>1:13 [347]8  41 39 30 39 35 38 37 37
560  638382.972<07>1:13 [347]8  42 20 30 20 20 20 20 20
561  638384.108<07>1:13 [347]8  43 20 20 20 20 20 20 20
562  638385.3  <07>1:13 [347]8  44 20 20 20 20 20 20 20
563  638386.276<07>1:13 [347]6  85 20 20 20 20 20
```

## Table C.2 The time delays of the Java application

| Index | Time (ms) | DeviceNet I.D.# | Data | Event Description | Time (ms) |
|---|---|---|---|---|---|
| 0 | 371513.67 <07>2:05 [43D]2 | 11 0 | | PLC trigger Bcode read | |
| 1 | 371515.689<07>1:15 [3C7]0 | | | | |
| 2 | 371780.069<07>1:13 [347]8 | 0 7 90 33 39 35 31 39 | | Barcode read complete | 266.399 |
| 3 | 371781.223<07>1:13 [347]8 | 41 39 30 39 35 38 37 37 | | | |
| 4 | 371782.455<07>1:13 [347]8 | 42 0 30 0 0 0 0 0 | | | |
| 5 | 371783.688<07>1:13 [347]8 | 43 0 0 0 0 0 0 0 | | | |
| 6 | 371784.92 <07>1:13 [347]8 | 44 0 0 0 0 0 0 0 | | | |
| 7 | 371786.016<07>1:13 [347]6 | 85 0 0 0 0 0 | | | 5.947 |
| 8 | 371788.209<07>2:02 [43A]0 | | | | |
| 9 | 371811.889<07>1:13 [347]8 | 0 7 90 33 39 35 31 39 | | | |
| 10 | 371813.105<07>1:13 [347]8 | 41 39 30 39 35 38 37 37 | | | |
| 11 | 371814.37 <07>1:13 [347]8 | 42 0 30 0 0 0 0 0 | | | |
| 12 | 371815.602<07>1:13 [347]8 | 43 0 0 0 0 0 0 0 | | | |
| 13 | 371816.834<07>1:13 [347]8 | 44 0 0 0 0 0 0 0 | | | |
| 14 | 371819.699<07>1:13 [347]6 | 85 0 0 0 0 0 | | | 7.81 |
| 15 | 371821.019<07>2:02 [43A]0 | | | | |
| 16 | 372443.534<07>2:05 [43D]2 | 10 0 | | | |
| 17 | 372445.54 <07>1:15 [3C7]0 | | | | |
| 18 | 372780.598<07>1:13 [347]8 | 0 6 90 33 39 35 31 39 | | | |
| 19 | 372781.822<07>1:13 [347]8 | 41 39 30 39 35 38 37 37 | | | |
| 20 | 372783.086<07>1:13 [347]8 | 42 0 30 0 0 0 0 0 | | | |
| 21 | 372784.319<07>1:13 [347]8 | 43 0 0 0 0 0 0 0 | | | |
| 22 | 372785.575<07>1:13 [347]8 | 44 0 0 0 0 0 0 0 | | | |
| 23 | 372786.664<07>1:13 [347]6 | 85 0 0 0 0 0 | | | 6.066 |
| 24 | 372788.687<07>2:02 [43A]0 | | | | |
| 25 | 372812.736<07>1:13 [347]8 | 0 6 90 33 39 35 31 39 | | | |
| 26 | 372813.952<07>1:13 [347]8 | 41 39 30 39 35 38 37 37 | | | |
| 27 | 372815.215<07>1:13 [347]8 | 42 0 30 0 0 0 0 0 | | | |
| 28 | 372816.447<07>1:13 [347]8 | 43 0 0 0 0 0 0 0 | | | |
| 29 | 372820.256<07>1:13 [347]8 | 44 0 0 0 0 0 0 0 | | | |
| 30 | 372821.32 <07>1:13 [347]6 | 85 0 0 0 0 0 | | | 8.584 |
| 31 | 372822.698<07>2:02 [43A]0 | | | | |

```
32   373196.113<07>2:05 [43D]2  12 0                                        PLC's read Ack
33   373200.769<07>1:15 [3C7]0
34   373330.653<07>1:13 [347]8  0  4  90 33 39 35 31 39
35   373331.855<07>1:13 [347]8  41 39 30 39 35 38 37 37
36   373333.118<07>1:13 [347]8  42 0  30 0  0  0  0  0
37   373334.352<07>1:13 [347]8  43 0  0  0  0  0  0  0
38   373335.582<07>1:13 [347]8  44 0  0  0  0  0  0  0
39   373336.687<07>1:13 [347]6  85 0  0  0  0  0                                                    6.034
40   373337.791<07>2:02 [43A]0
41   373340.719<07>1:13 [347]8  0  0  90 33 39 35 31 39   Ack acked                               144.606
42   373341.895<07>1:13 [347]8  41 39 30 39 35 38 37 37
43   373343.127<07>1:13 [347]8  42 0  30 0  0  0  0  0
44   373344.359<07>1:13 [347]8  43 0  0  0  0  0  0  0
45   373345.592<07>1:13 [347]8  44 0  0  0  0  0  0  0
46   373346.696<07>1:13 [347]6  85 0  0  0  0  0                                                    5.977
47   373347.857<07>2:02 [43A]0
48   373373.192<07>1:13 [347]8  0  0  90 33 39 35 31 39
49   373374.385<07>1:13 [347]8  41 39 30 39 35 38 37 37
50   373375.657<07>1:13 [347]8  42 0  30 0  0  0  0  0
51   373376.889<07>1:13 [347]8  43 0  0  0  0  0  0  0
52   373380.473<07>1:13 [347]8  44 0  0  0  0  0  0  0
53   373381.546<07>1:13 [347]6  85 0  0  0  0  0                                                    8.354
54   373382.745<07>2:02 [43A]0
55   374070.874<07>2:05 [43D]2  10 0
56   374072.884<07>1:15 [3C7]0
57   374852.89 <07>2:05 [43D]2  11 0                                        PLC trigger Bcode read
58   374854.913<07>1:15 [3C7]0
59   375041.328<07>1:13 [347]8  0  1  90 33 39 35 31 39
60   375042.505<07>1:13 [347]8  41 39 30 39 35 38 37 37
61   375043.737<07>1:13 [347]8  42 0  30 0  0  0  0  0
62   375044.969<07>1:13 [347]8  43 0  0  0  0  0  0  0
63   375046.199<07>1:13 [347]8  44 0  0  0  0  0  0  0
64   375047.304<07>1:13 [347]6  85 0  0  0  0  0                                                    5.976
65   375048.537<07>2:02 [43A]0
66   375073.633<07>1:13 [347]8  0  1  90 33 39 35 31 39
67   375074.809<07>1:13 [347]8  41 39 30 39 35 38 37 37
68   375076.065<07>1:13 [347]8  42 0  30 0  0  0  0  0
69   375077.298<07>1:13 [347]8  43 0  0  0  0  0  0  0
70   375081.195<07>1:13 [347]8  44 0  0  0  0  0  0  0
71   375082.275<07>1:13 [347]6  85 0  0  0  0  0                                                    8.642
72   375083.907<07>2:02 [43A]0
73   375211.382<07>1:13 [347]8  0  7  90 33 39 35 31 39   Barcode read complete                   358.492
74   375212.606<07>1:13 [347]8  41 39 30 39 35 38 37 37
75   375213.871<07>1:13 [347]8  42 0  30 0  0  0  0  0
76   375215.102<07>1:13 [347]8  43 0  0  0  0  0  0  0
77   375216.335<07>1:13 [347]8  44 0  0  0  0  0  0  0
78   375217.44 <07>1:13 [347]6  85 0  0  0  0  0                                                    6.058
79   375219.64 <07>2:02 [43A]0
80   375243.768<07>1:13 [347]8  0  7  90 33 39 35 31 39
81   375244.945<07>1:13 [347]8  41 39 30 39 35 38 37 37
82   375246.201<07>1:13 [347]8  42 0  30 0  0  0  0  0
83   375247.433<07>1:13 [347]8  43 0  0  0  0  0  0  0
84   375251.203<07>1:13 [347]8  44 0  0  0  0  0  0  0
85   375252.299<07>1:13 [347]6  85 0  0  0  0  0                                                    8.531
86   375253.474<07>2:02 [43A]0
87   375699.698<07>2:05 [43D]2  10 0
88   375702.156<07>1:15 [3C7]0
89   376041.71 <07>1:13 [347]8  0  6  90 33 39 35 31 39
90   376042.903<07>1:13 [347]8  41 39 30 39 35 38 37 37
91   376044.176<07>1:13 [347]8  42 0  30 0  0  0  0  0
92   376045.408<07>1:13 [347]8  43 0  0  0  0  0  0  0
93   376046.64 <07>1:13 [347]8  44 0  0  0  0  0  0  0
94   376047.743<07>1:13 [347]6  85 0  0  0  0  0
95   376048.808<07>2:02 [43A]0
96   376073.472<07>1:13 [347]8  0  6  90 33 39 35 31 39
97   376074.648<07>1:13 [347]8  41 39 30 39 35 38 37 37
98   376075.896<07>1:13 [347]8  42 0  30 0  0  0  0  0
99   376077.128<07>1:13 [347]8  43 0  0  0  0  0  0  0
100  376078.425<07>1:13 [347]8  44 0  0  0  0  0  0  0
101  376081.354<07>1:13 [347]6  85 0  0  0  0  0
102  376082.521<07>2:02 [43A]0
103  376451.015<07>2:05 [43D]2  12 0                                        PLC's read Ack
104  376453.032<07>1:15 [3C7]0
105  376591.919<07>1:13 [347]8  0  4  90 33 39 35 31 39
106  376593.135<07>1:13 [347]8  41 39 30 39 35 38 37 37
107  376594.407<07>1:13 [347]8  42 0  30 0  0  0  0  0
108  376595.64 <07>1:13 [347]8  43 0  0  0  0  0  0  0
109  376596.872<07>1:13 [347]8  44 0  0  0  0  0  0  0
110  376597.976<07>1:13 [347]6  85 0  0  0  0  0
111  376599.256<07>2:02 [43A]0
112  376601.993<07>1:13 [347]8  0  0  90 33 39 35 31 39   Ack acked                 150.978
113  376603.171<07>1:13 [347]8  41 39 30 39 35 38 37 37
114  376604.41 <07>1:13 [347]8  42 0  30 0  0  0  0  0
115  376605.643<07>1:13 [347]8  43 0  0  0  0  0  0  0
116  376606.875<07>1:13 [347]8  44 0  0  0  0  0  0  0
```

```
117  376607.977<07>1:13 [347]6  85 0  0  0  0  0
118  376609.755<07>2:02 [43A]0
119  376632.93 <07>1:13 [347]8  0  0  90 33 39 35 31 39
120  376634.106<07>1:13 [347]8  41 39 30 39 35 38 37 37
121  376635.337<07>1:13 [347]8  42 0  30 0  0  0  0  0
122  376636.569<07>1:13 [347]8  43 0  0  0  0  0  0  0
123  376637.852<07>1:13 [347]8  44 0  0  0  0  0  0  0
124  376641.572<07>1:13 [347]6  85 0  0  0  0  0
125  376642.86 <07>2:02 [43A]0
126  377292.546<07>2:05 [43D]2  10 0
127  377294.555<07>1:15 [3C7]0
128  378043.962<07>2:05 [43D]2  11 0                                  PLC trigger Bcode read
129  378045.987<07>1:15 [3C7]0
130  378262.58 <07>1:13 [347]8  0  1  90 33 39 35 31 39
131  378263.756<07>1:13 [347]8  41 39 30 39 35 38 37 37
132  378265.006<07>1:13 [347]8  42 0  30 0  0  0  0  0
133  378266.236<07>1:13 [347]8  43 0  0  0  0  0  0  0
134  378267.469<07>1:13 [347]8  44 0  0  0  0  0  0  0
135  378268.574<07>1:13 [347]6  85 0  0  0  0  0
136  378269.709<07>2:02 [43A]0
137  378294.351<07>1:13 [347]8  0  1  90 33 39 35 31 39
138  378295.582<07>1:13 [347]8  41 39 30 39 35 38 37 37
139  378296.828<07>1:13 [347]8  42 0  30 0  0  0  0  0
140  378298.061<07>1:13 [347]8  43 0  0  0  0  0  0  0
141  378299.293<07>1:13 [347]8  44 0  0  0  0  0  0  0
142  378302.28 <07>1:13 [347]6  85 0  0  0  0  0
143  378303.455<07>2:02 [43A]0
144  378422.631<07>1:13 [347]8  0  7  90 33 39 35 31 39   Barcode read complete    378.669
145  378423.807<07>1:13 [347]8  41 39 30 39 35 38 37 37
146  378425.048<07>1:13 [347]8  42 0  30 0  0  0  0  0
147  378426.279<07>1:13 [347]8  43 0  0  0  0  0  0  0
148  378427.513<07>1:13 [347]8  44 0  0  0  0  0  0  0
149  378428.616<07>1:13 [347]6  85 0  0  0  0  0
150  378430.104<07>2:02 [43A]0
151  378454.457<07>1:13 [347]8  0  7  90 33 39 35 31 39
152  378455.633<07>1:13 [347]8  41 39 30 39 35 38 37 37
153  378456.874<07>1:13 [347]8  42 0  30 0  0  0  0  0
154  378458.105<07>1:13 [347]8  43 0  0  0  0  0  0  0
155  378459.337<07>1:13 [347]8  44 0  0  0  0  0  0  0
156  378462.281<07>1:13 [347]6  85 0  0  0  0  0
157  378463.484<07>2:02 [43A]0
158  378884.347<07>2:05 [43D]2  10 0
159  378886.357<07>1:15 [3C7]0
160  379262.972<07>1:13 [347]8  0  6  90 33 39 35 31 39
161  379264.182<07>1:13 [347]8  41 39 30 39 35 38 37 37
162  379265.444<07>1:13 [347]8  42 0  30 0  0  0  0  0
163  379266.676<07>1:13 [347]8  43 0  0  0  0  0  0  0
164  379267.908<07>1:13 [347]8  44 0  0  0  0  0  0  0
165  379269.021<07>1:13 [347]6  85 0  0  0  0  0
166  379270.244<07>2:02 [43A]0
167  379294.172<07>1:13 [347]8  0  6  90 33 39 35 31 39
168  379295.398<07>1:13 [347]8  41 39 30 39 35 38 37 37
169  379296.669<07>1:13 [347]8  42 0  30 0  0  0  0  0
170  379297.901<07>1:13 [347]8  43 0  0  0  0  0  0  0
171  379299.134<07>1:13 [347]8  44 0  0  0  0  0  0  0
172  379302.606<07>1:13 [347]6  85 0  0  0  0  0
173  379304.621<07>2:02 [43A]0
174  379674.91 <07>2:05 [43D]2  12 0                                  PLC's read Ack
175  379676.934<07>1:15 [3C7]0
176  379803.168<07>1:13 [347]8  0  4  90 33 39 35 31 39
177  379804.344<07>1:13 [347]8  41 39 30 39 35 38 37 37
178  379805.576<07>1:13 [347]8  42 0  30 0  0  0  0  0
179  379806.808<07>1:13 [347]8  43 0  0  0  0  0  0  0
180  379808.04 <07>1:13 [347]8  44 0  0  0  0  0  0  0
181  379809.144<07>1:13 [347]6  85 0  0  0  0  0
182  379810.456<07>2:02 [43A]0
183  379813.241<07>1:13 [347]8  0  0  90 33 39 35 31 39   Ack acked                138.331
184  379814.417<07>1:13 [347]8  41 39 30 39 35 38 37 37
185  379815.681<07>1:13 [347]8  42 0  30 0  0  0  0  0
186  379816.913<07>1:13 [347]8  43 0  0  0  0  0  0  0
187  379818.146<07>1:13 [347]8  44 0  0  0  0  0  0  0
188  379819.249<07>1:13 [347]6  85 0  0  0  0  0
189  379821.402<07>2:02 [43A]0
190  379845.626<07>1:13 [347]8  0  0  90 33 39 35 31 39
191  379846.844<07>1:13 [347]8  41 39 30 39 35 38 37 37
192  379848.115<07>1:13 [347]8  42 0  30 0  0  0  0  0
193  379849.349<07>1:13 [347]8  43 0  0  0  0  0  0  0
194  379852.989<07>1:13 [347]8  44 0  0  0  0  0  0  0
195  379854.052<07>1:13 [347]6  85 0  0  0  0  0
196  379856.133<07>2:02 [43A]0
197  380513.725<07>2:05 [43D]2  10 0
198  380515.735<07>1:15 [3C7]0
199  381235.703<07>2:05 [43D]2  11 0                                  PLC trigger Bcode read
200  381237.737<07>1:15 [3C7]0
201  381493.837<07>1:13 [347]8  0  1  90 33 39 35 31 39
```

```
202  381495.013<07>1:13 [347]8  41 39 30 39 35 38 37 37
203  381496.286<07>1:13 [347]8  42 0  30 0  0  0  0  0
204  381497.518<07>1:13 [347]8  43 0  0  0  0  0  0  0
205  381498.75 <07>1:13 [347]8  44 0  0  0  0  0  0  0
206  381499.855<07>1:13 [347]6  85 0  0  0  0  0
207  381501.014<07>2:02 [43A]0
208  381525.038<07>1:13 [347]8  0  1  90 33 39 35 31 39
209  381526.241<07>1:13 [347]8  41 39 30 39 35 38 37 37
210  381527.504<07>1:13 [347]8  42 0  30 0  0  0  0  0
211  381528.735<07>1:13 [347]8  43 0  0  0  0  0  0  0
212  381529.967<07>1:13 [347]8  44 0  0  0  0  0  0  0
213  381533.48 <07>1:13 [347]6  85 0  0  0  0  0
214  381535.424<07>2:02 [43A]0
215  381674.055<07>1:13 [347]8  0  7  90 33 39 35 31 39   Barcode read complete     438.352
216  381675.264<07>1:13 [347]8  41 39 30 39 35 38 37 37
217  381676.528<07>1:13 [347]8  42 0  30 0  0  0  0  0
218  381677.762<07>1:13 [347]8  43 0  0  0  0  0  0  0
219  381678.993<07>1:13 [347]8  44 0  0  0  0  0  0  0
220  381680.096<07>1:13 [347]6  85 0  0  0  0  0
221  381681.417<07>2:02 [43A]0
222  381706.184<07>1:13 [347]8  0  7  90 33 39 35 31 39
223  381707.393<07>1:13 [347]8  41 39 30 39 35 38 37 37
224  381708.657<07>1:13 [347]8  42 0  30 0  0  0  0  0
225  381709.89 <07>1:13 [347]8  43 0  0  0  0  0  0  0
226  381713.715<07>1:13 [347]8  44 0  0  0  0  0  0  0
227  381714.779<07>1:13 [347]6  85 0  0  0  0  0
228  381717.178<07>2:02 [43A]0
229  382112.184<07>2:05 [43D]2  10 0
230  382114.648<07>1:15 [3C7]0
231  382514.231<07>1:13 [347]8  0  6  90 33 39 35 31 39
232  382515.407<07>1:13 [347]8  41 39 30 39 35 38 37 37
233  382516.639<07>1:13 [347]8  42 0  30 0  0  0  0  0
234  382517.871<07>1:13 [347]8  43 0  0  0  0  0  0  0
235  382519.104<07>1:13 [347]8  44 0  0  0  0  0  0  0
236  382520.201<07>1:13 [347]6  85 0  0  0  0  0
237  382521.543<07>2:02 [43A]0
238  382545.895<07>1:13 [347]8  0  6  90 33 39 35 31 39
239  382547.087<07>1:13 [347]8  41 39 30 39 35 38 37 37
240  382548.353<07>1:13 [347]8  42 0  30 0  0  0  0  0
241  382549.584<07>1:13 [347]8  43 0  0  0  0  0  0  0
242  382550.816<07>1:13 [347]8  44 0  0  0  0  0  0  0
243  382553.873<07>1:13 [347]6  85 0  0  0  0  0
244  382555.194<07>2:02 [43A]0
245  382834.179<07>2:05 [43D]2  12 0                     PLC's read Ack
246  382836.189<07>1:15 [3C7]0
247  383054.439<07>1:13 [347]8  0  4  90 33 39 35 31 39
248  383055.613<07>1:13 [347]8  41 39 30 39 35 38 37 37
249  383056.862<07>1:13 [347]8  42 0  30 0  0  0  0  0
250  383058.094<07>1:13 [347]8  43 0  0  0  0  0  0  0
251  383059.326<07>1:13 [347]8  44 0  0  0  0  0  0  0
252  383060.431<07>1:13 [347]6  85 0  0  0  0  0
253  383061.463<07>2:02 [43A]0
254  383064.503<07>1:13 [347]8  0  0  90 33 39 35 31 39   Ack acked                 230.324
255  383065.695<07>1:13 [347]8  41 39 30 39 35 38 37 37
256  383066.961<07>1:13 [347]8  42 0  30 0  0  0  0  0
257  383068.192<07>1:13 [347]8  43 0  0  0  0  0  0  0
258  383069.424<07>1:13 [347]8  44 0  0  0  0  0  0  0
259  383070.528<07>1:13 [347]6  85 0  0  0  0  0
260  383071.656<07>2:02 [43A]0
261  383096.351<07>1:13 [347]8  0  0  90 33 39 35 31 39
262  383097.528<07>1:13 [347]8  41 39 30 39 35 38 37 37
263  383098.784<07>1:13 [347]8  42 0  30 0  0  0  0  0
264  383100.016<07>1:13 [347]8  43 0  0  0  0  0  0  0
265  383101.312<07>1:13 [347]8  44 0  0  0  0  0  0  0
266  383104.274<07>1:13 [347]6  85 0  0  0  0  0
267  383105.45 <07>2:02 [43A]0
268  383734.516<07>2:05 [43D]2  10 0
269  383736.526<07>1:15 [3C7]0
270  384516.162<07>2:05 [43D]2  11 0                     PLC trigger Bcode read
271  384518.212<07>1:15 [3C7]0
272  384865.147<07>1:13 [347]8  0  1  90 33 39 35 31 39
273  384866.33 <07>1:13 [347]8  41 39 30 39 35 38 37 37
274  384867.594<07>1:13 [347]8  42 0  30 0  0  0  0  0
275  384868.826<07>1:13 [347]8  43 0  0  0  0  0  0  0
276  384870.07 <07>1:13 [347]8  44 0  0  0  0  0  0  0
277  384871.163<07>1:13 [347]6  85 0  0  0  0  0
278  384873.732<07>2:02 [43A]0
279  384896.867<07>1:13 [347]8  0  1  90 33 39 35 31 39
280  384898.052<07>1:13 [347]8  41 39 30 39 35 38 37 37
281  384899.323<07>1:13 [347]8  42 0  30 0  0  0  0  0
282  384900.554<07>1:13 [347]8  43 0  0  0  0  0  0  0
283  384901.787<07>1:13 [347]8  44 0  0  0  0  0  0  0
284  384904.812<07>1:13 [347]6  85 0  0  0  0  0
285  384906.861<07>2:02 [43A]0
286  385035.207<07>1:13 [347]8  0  7  90 33 39 35 31 39   Barcode read complete     519.045
```

```
287  385036.383<07>1:13 [347]8  41 39 30 39 35 38 37 37
288  385037.632<07>1:13 [347]8  42 0  30 0  0  0  0  0
289  385038.952<07>1:13 [347]8  43 0  0  0  0  0  0  0
290  385040.185<07>1:13 [347]8  44 0  0  0  0  0  0  0
291  385041.249<07>1:13 [347]6  85 0  0  0  0  0
292  385043.242<07>2:02 [43A]0
293  385067.002<07>1:13 [347]8  0  7  90 33 39 35 31 39
294  385068.196<07>1:13 [347]8  41 39 30 39 35 38 37 37
295  385069.458<07>1:13 [347]8  42 0  30 0  0  0  0  0
296  385070.69 <07>1:13 [347]8  43 0  0  0  0  0  0  0
297  385071.924<07>1:13 [347]8  44 0  0  0  0  0  0  0
298  385074.85 <07>1:13 [347]6  85 0  0  0  0  0
299  385076.17 <07>2:02 [43A]0
300  385537.87 <07>2:05 [43D]2  10 0
301  385539.879<07>1:15 [3C7]0
302  385875.544<07>1:13 [347]8  0  6  90 33 39 35 31 39
303  385876.768<07>1:13 [347]8  41 39 30 39 35 38 37 37
304  385878.04 <07>1:13 [347]8  42 0  30 0  0  0  0  0
305  385879.272<07>1:13 [347]8  43 0  0  0  0  0  0  0
306  385880.505<07>1:13 [347]8  44 0  0  0  0  0  0  0
307  385881.608<07>1:13 [347]6  85 0  0  0  0  0
308  385882.761<07>2:02 [43A]0
309  385906.713<07>1:13 [347]8  0  6  90 33 39 35 31 39
310  385907.889<07>1:13 [347]8  41 39 30 39 35 38 37 37
311  385909.154<07>1:13 [347]8  42 0  30 0  0  0  0  0
312  385910.385<07>1:13 [347]8  43 0  0  0  0  0  0  0
313  385911.619<07>1:13 [347]8  44 0  0  0  0  0  0  0
314  385915.179<07>1:13 [347]6  85 0  0  0  0  0
315  385916.355<07>2:02 [43A]0
316  386232.51 <07>2:05 [43D]2  12 0                          PLC's read Ack
317  386236.255<07>1:15 [3C7]0
318  386425.742<07>1:13 [347]8  0  0  90 33 39 35 31 39   Ack acked                  193.232
319  386426.917<07>1:13 [347]8  41 39 30 39 35 38 37 37
320  386428.168<07>1:13 [347]8  42 0  30 0  0  0  0  0
321  386429.399<07>1:13 [347]8  43 0  0  0  0  0  0  0
322  386430.63 <07>1:13 [347]8  44 0  0  0  0  0  0  0
323  386431.745<07>1:13 [347]6  85 0  0  0  0  0
324  386432.959<07>2:02 [43A]0
325  386458.169<07>1:13 [347]8  0  0  90 33 39 35 31 39
326  386459.344<07>1:13 [347]8  41 39 30 39 35 38 37 37
327  386460.601<07>1:13 [347]8  42 0  30 0  0  0  0  0
328  386461.832<07>1:13 [347]8  43 0  0  0  0  0  0  0
329  386465.562<07>1:13 [347]8  44 0  0  0  0  0  0  0
330  386466.627<07>1:13 [347]6  85 0  0  0  0  0
331  386467.89 <07>2:02 [43A]0
332  387651.145<07>2:05 [43D]2  10 0
333  387656.016<07>1:15 [3C7]0
334  389219.997<07>2:05 [43D]2  11 0                          PLC trigger Bcode read
335  389222.068<07>1:15 [3C7]0
336  389787.062<07>1:13 [347]8  0  1  90 33 39 35 31 39
337  389788.236<07>1:13 [347]8  41 39 30 39 35 38 37 37
338  389789.476<07>1:13 [347]8  42 0  30 0  0  0  0  0
339  389790.709<07>1:13 [347]8  43 0  0  0  0  0  0  0
340  389791.94 <07>1:13 [347]8  44 0  0  0  0  0  0  0
341  389793.045<07>1:13 [347]6  85 0  0  0  0  0
342  389794.197<07>2:02 [43A]0
343  389819.013<07>1:13 [347]8  0  1  90 33 39 35 31 39
344  389820.237<07>1:13 [347]8  41 39 30 39 35 38 37 37
345  389821.51 <07>1:13 [347]8  42 0  30 0  0  0  0  0
346  389822.742<07>1:13 [347]8  43 0  0  0  0  0  0  0
347  389826.832<07>1:13 [347]8  44 0  0  0  0  0  0  0
348  389827.904<07>1:13 [347]6  85 0  0  0  0  0
349  389829.153<07>2:02 [43A]0
350  389947.12 <07>1:13 [347]8  0  7  90 33 39 35 31 39   Barcode read complete     727.123
351  389948.296<07>1:13 [347]8  41 39 30 39 35 38 37 37
352  389949.528<07>1:13 [347]8  42 0  30 0  0  0  0  0
353  389950.76 <07>1:13 [347]8  43 0  0  0  0  0  0  0
354  389952.041<07>1:13 [347]8  44 0  0  0  0  0  0  0
355  389953.104<07>1:13 [347]6  85 0  0  0  0  0
356  389954.624<07>2:02 [43A]0
357  389978.12 <07>1:13 [347]8  0  7  90 33 39 35 31 39
358  389979.296<07>1:13 [347]8  41 39 30 39 35 38 37 37
359  389980.538<07>1:13 [347]8  42 0  30 0  0  0  0  0
360  389981.77 <07>1:13 [347]8  43 0  0  0  0  0  0  0
361  389983.003<07>1:13 [347]8  44 0  0  0  0  0  0  0
362  389986.763<07>1:13 [347]6  85 0  0  0  0  0
363  389988.819<07>2:02 [43A]0
364  390394.386<07>2:05 [43D]2  10 0
365  390397.876<07>1:15 [3C7]0
366  390787.456<07>1:13 [347]8  0  6  90 33 39 35 31 39
367  390788.751<07>1:13 [347]8  41 39 30 39 35 38 37 37
368  390790.024<07>1:13 [347]8  42 0  30 0  0  0  0  0
369  390791.256<07>1:13 [347]8  43 0  0  0  0  0  0  0
370  390792.487<07>1:13 [347]8  44 0  0  0  0  0  0  0
371  390793.592<07>1:13 [347]6  85 0  0  0  0  0
```

```
372  390794.912<07>2:02 [43A]0
373  390819.832<07>1:13 [347]8  0  6  90 33 39 35 31 39
374  390821.009<07>1:13 [347]8  41 39 30 39 35 38 37 37
375  390822.251<07>1:13 [347]8  42 0  30 0  0  0  0  0
376  390823.481<07>1:13 [347]8  43 0  0  0  0  0  0  0
377  390827.267<07>1:13 [347]8  44 0  0  0  0  0  0  0
378  390828.345<07>1:13 [347]6  85 0  0  0  0  0
379  390829.587<07>2:02 [43A]0
380  391179.109<07>2:05 [43D]2  12 0                                  PLC's read Ack
381  391181.117<07>1:15 [3C7]0
382  391327.653<07>1:13 [347]8  0  4  90 33 39 35 31 39
383  391328.83 <07>1:13 [347]8  41 39 30 39 35 38 37 37
384  391330.069<07>1:13 [347]8  42 0  30 0  0  0  0  0
385  391331.294<07>1:13 [347]8  43 0  0  0  0  0  0  0
386  391332.574<07>1:13 [347]8  44 0  0  0  0  0  0  0
387  391333.639<07>1:13 [347]6  85 0  0  0  0  0
388  391335.896<07>2:02 [43A]0
389  391337.719<07>1:13 [347]8  0  0  90 33 39 35 31 39   Ack acked                  158.61
390  391338.897<07>1:13 [347]8  41 39 30 39 35 38 37 37
391  391340.257<07>1:13 [347]8  42 0  30 0  0  0  0  0
392  391341.488<07>1:13 [347]8  43 0  0  0  0  0  0  0
393  391342.72 <07>1:13 [347]8  44 0  0  0  0  0  0  0
394  391343.826<07>1:13 [347]6  85 0  0  0  0  0
395  391345.299<07>2:02 [43A]0
396  391369.289<07>1:13 [347]8  0  0  90 33 39 35 31 39
397  391370.504<07>1:13 [347]8  41 39 30 39 35 38 37 37
398  391371.776<07>1:13 [347]8  42 0  30 0  0  0  0  0
399  391373.008<07>1:13 [347]8  43 0  0  0  0  0  0  0
400  391374.241<07>1:13 [347]8  44 0  0  0  0  0  0  0
401  391377.308<07>1:13 [347]6  85 0  0  0  0  0
402  391378.362<07>2:02 [43A]0
403  392143.044<07>2:05 [43D]2  10 0
404  392147.894<07>1:15 [3C7]0
405  392956.211<07>2:05 [43D]2  11 0                                  PLC trigger Bcode read
406  392958.892<07>1:15 [3C7]0
407  393528.517<07>1:13 [347]8  0  1  90 33 39 35 31 39
408  393529.709<07>1:13 [347]8  41 39 30 39 35 38 37 37
409  393530.979<07>1:13 [347]8  42 0  30 0  0  0  0  0
410  393532.211<07>1:13 [347]8  43 0  0  0  0  0  0  0
411  393533.444<07>1:13 [347]8  44 0  0  0  0  0  0  0
412  393534.548<07>1:13 [347]6  85 0  0  0  0  0
413  393535.726<07>2:02 [43A]0
414  393560.135<07>1:13 [347]8  0  1  90 33 39 35 31 39
415  393561.334<07>1:13 [347]8  41 39 30 39 35 38 37 37
416  393562.607<07>1:13 [347]8  42 0  30 0  0  0  0  0
417  393563.839<07>1:13 [347]8  43 0  0  0  0  0  0  0
418  393565.07 <07>1:13 [347]8  44 0  0  0  0  0  0  0
419  393568.174<07>1:13 [347]6  85 0  0  0  0  0
420  393569.43 <07>2:02 [43A]0
421  393698.578<07>1:13 [347]8  0  7  90 33 39 35 31 39   Barcode read complete     742.367
422  393699.754<07>1:13 [347]8  41 39 30 39 35 38 37 37
423  393701.021<07>1:13 [347]8  42 0  30 0  0  0  0  0
424  393702.252<07>1:13 [347]8  43 0  0  0  0  0  0  0
425  393703.483<07>1:13 [347]8  44 0  0  0  0  0  0  0
426  393704.587<07>1:13 [347]6  85 0  0  0  0  0
427  393705.731<07>2:02 [43A]0
428  393730.267<07>1:13 [347]8  0  7  90 33 39 35 31 39
429  393731.475<07>1:13 [347]8  41 39 30 39 35 38 37 37
430  393732.747<07>1:13 [347]8  42 0  30 0  0  0  0  0
431  393733.98 <07>1:13 [347]8  43 0  0  0  0  0  0  0
432  393735.212<07>1:13 [347]8  44 0  0  0  0  0  0  0
433  393738.221<07>1:13 [347]6  85 0  0  0  0  0
434  393739.542<07>2:02 [43A]0
435  394528.905<07>1:13 [347]8  0  6  90 33 39 35 31 39
436  394530.081<07>1:13 [347]8  41 39 30 39 35 38 37 37
437  394531.321<07>1:13 [347]8  42 0  30 0  0  0  0  0
438  394532.554<07>1:13 [347]8  43 0  0  0  0  0  0  0
439  394533.785<07>1:13 [347]8  44 0  0  0  0  0  0  0
440  394534.892<07>1:13 [347]6  85 0  0  0  0  0
441  394536.003<07>2:02 [43A]0
442  394549.068<07>2:05 [43D]2  10 0
443  394551.093<07>1:15 [3C7]0
444  394563.978<07>1:13 [347]8  0  6  90 33 39 35 31 39
445  394565.154<07>1:13 [347]8  41 39 30 39 35 38 37 37
446  394568.715<07>1:13 [347]8  42 0  30 0  0  0  0  0
447  394569.948<07>1:13 [347]8  43 0  0  0  0  0  0  0
448  394571.227<07>1:13 [347]8  44 0  0  0  0  0  0  0
449  394572.292<07>1:13 [347]6  85 0  0  0  0  0
450  394573.691<07>2:02 [43A]0
451  395688.744<07>2:05 [43D]2  12 0                                  PLC's read Ack
452  395690.76 <07>1:15 [3C7]0
453  396149.532<07>1:13 [347]8  0  0  90 33 39 35 31 39   Ack acked                  460.788
454  396150.74 <07>1:13 [347]8  41 39 30 39 35 38 37 37
455  396152.013<07>1:13 [347]8  42 0  30 0  0  0  0  0
456  396153.245<07>1:13 [347]8  43 0  0  0  0  0  0  0
```

```
457  396154.477<07>1:13 [347]8  44 0  0  0  0  0  0  0
458  396155.58 <07>1:13 [347]6  85 0  0  0  0  0
459  396156.693<07>2:02 [43A]0
460  396181.325<07>1:13 [347]8  0  0  90 33 39 35 31 39
461  396182.503<07>1:13 [347]8  41 39 30 39 35 38 37 37
462  396183.734<07>1:13 [347]8  42 0  30 0  0  0  0  0
463  396184.965<07>1:13 [347]8  43 0  0  0  0  0  0  0
464  396186.199<07>1:13 [347]8  44 0  0  0  0  0  0  0
465  396189.176<07>1:13 [347]6  85 0  0  0  0  0
466  396190.495<07>2:02 [43A]0
467  396559.35 <07>2:05 [43D]2  10 0
468  396561.375<07>1:15 [3C7]0
469  397759.189<07>2:05 [43D]2  11 0                                  PLC trigger Bcode read
470  397761.214<07>1:15 [3C7]0
471  398320.378<07>1:13 [347]8  0  1  90 33 39 35 31 39
472  398321.563<07>1:13 [347]8  41 39 30 39 35 38 37 37
473  398322.836<07>1:13 [347]8  42 0  30 0  0  0  0  0
474  398324.068<07>1:13 [347]8  43 0  0  0  0  0  0  0
475  398325.301<07>1:13 [347]8  44 0  0  0  0  0  0  0
476  398326.404<07>1:13 [347]6  85 0  0  0  0  0
477  398327.549<07>2:02 [43A]0
478  398352.174<07>1:13 [347]8  0  1  90 33 39 35 31 39
479  398353.388<07>1:13 [347]8  41 39 30 39 35 38 37 37
480  398354.661<07>1:13 [347]8  42 0  30 0  0  0  0  0
481  398355.892<07>1:13 [347]8  43 0  0  0  0  0  0  0
482  398357.149<07>1:13 [347]8  44 0  0  0  0  0  0  0
483  398360.022<07>1:13 [347]6  85 0  0  0  0  0
484  398361.214<07>2:02 [43A]0
485  398480.439<07>1:13 [347]8  0  7  90 33 39 35 31 39  Barcode read complete  721.25
486  398481.614<07>1:13 [347]8  41 39 30 39 35 38 37 37
487  398482.872<07>1:13 [347]8  42 0  30 0  0  0  0  0
488  398484.103<07>1:13 [347]8  43 0  0  0  0  0  0  0
489  398485.337<07>1:13 [347]8  44 0  0  0  0  0  0  0
490  398486.439<07>1:13 [347]6  85 0  0  0  0  0
491  398487.44 <07>2:02 [43A]0
492  398512.279<07>1:13 [347]8  0  7  90 33 39 35 31 39
493  398513.456<07>1:13 [347]8  41 39 30 39 35 38 37 37
494  398514.696<07>1:13 [347]8  42 0  30 0  0  0  0  0
495  398515.927<07>1:13 [347]8  43 0  0  0  0  0  0  0
496  398517.16 <07>1:13 [347]8  44 0  0  0  0  0  0  0
497  398520.08 <07>1:13 [347]6  85 0  0  0  0  0
498  398521.08 <07>2:02 [43A]0
499  398905.298<07>2:05 [43D]2  10 0
500  398910.132<07>1:15 [3C7]0
501  399310.764<07>1:13 [347]8  0  6  90 33 39 35 31 39
502  399311.939<07>1:13 [347]8  41 39 30 39 35 38 37 37
503  399313.172<07>1:13 [347]8  42 0  30 0  0  0  0  0
504  399314.404<07>1:13 [347]8  43 0  0  0  0  0  0  0
505  399315.66 <07>1:13 [347]8  44 0  0  0  0  0  0  0
506  399316.74 <07>1:13 [347]6  85 0  0  0  0  0
507  399319.9  <07>2:02 [43A]0
508  401696.152<07>2:05 [43D]2  12 0                                  PLC's read Ack
509  401700.809<07>1:15 [3C7]0
510  401951.789<07>1:13 [347]8  0  0  90 33 39 35 31 39  Ack acked  255.637
511  401952.964<07>1:13 [347]8  41 39 30 39 35 38 37 37
512  401954.205<07>1:13 [347]8  42 0  30 0  0  0  0  0
513  401955.437<07>1:13 [347]8  43 0  0  0  0  0  0  0
514  401956.67 <07>1:13 [347]8  44 0  0  0  0  0  0  0
515  401957.775<07>1:13 [347]6  85 0  0  0  0  0
516  401958.919<07>2:02 [43A]0
517  401984.191<07>1:13 [347]8  0  0  90 33 39 35 31 39
518  401985.366<07>1:13 [347]8  41 39 30 39 35 38 37 37
519  401986.638<07>1:13 [347]8  42 0  30 0  0  0  0  0
520  401987.871<07>1:13 [347]8  43 0  0  0  0  0  0  0
521  401991.599<07>1:13 [347]8  44 0  0  0  0  0  0  0
522  401992.664<07>1:13 [347]6  85 0  0  0  0  0
523  401993.752<07>2:02 [43A]0
524  404263.585<07>2:05 [43D]2  10 0
525  404265.642<07>1:15 [3C7]0
526  405253.642<07>2:05 [43D]2  11 0                                  PLC trigger Bcode read
527  405255.698<07>1:15 [3C7]0
528  405683.243<07>1:13 [347]8  0  1  90 33 39 35 31 39
529  405684.435<07>1:13 [347]8  41 39 30 39 35 38 37 37
530  405685.7  <07>1:13 [347]8  42 0  30 0  0  0  0  0
531  405686.932<07>1:13 [347]8  43 0  0  0  0  0  0  0
532  405688.163<07>1:13 [347]8  44 0  0  0  0  0  0  0
533  405689.275<07>1:13 [347]6  85 0  0  0  0  0
534  405690.452<07>2:02 [43A]0
535  405714.349<07>1:13 [347]8  0  1  90 33 39 35 31 39
536  405715.556<07>1:13 [347]8  41 39 30 39 35 38 37 37
537  405716.82 <07>1:13 [347]8  42 0  30 0  0  0  0  0
538  405718.053<07>1:13 [347]8  43 0  0  0  0  0  0  0
539  405719.31 <07>1:13 [347]8  44 0  0  0  0  0  0  0
540  405722.926<07>1:13 [347]6  85 0  0  0  0  0
541  405724.102<07>2:02 [43A]0
```

```
542  405843.303<07>1:13 [347]8  0  7  90 33 39 35 31 39   Barcode read complete    589.661
543  405844.479<07>1:13 [347]8  41 39 30 39 35 38 37 37
544  405845.742<07>1:13 [347]8  42 0  30 0  0  0  0  0
545  405846.974<07>1:13 [347]8  43 0  0  0  0  0  0  0
546  405848.206<07>1:13 [347]8  44 0  0  0  0  0  0  0
547  405849.31 <07>1:13 [347]6  85 0  0  0  0  0
548  405851.488<07>2:02 [43A]0
549  405874.455<07>1:13 [347]8  0  7  90 33 39 35 31 39
550  405875.631<07>1:13 [347]8  41 39 30 39 35 38 37 37
551  405876.863<07>1:13 [347]8  42 0  30 0  0  0  0  0
552  405878.094<07>1:13 [347]8  43 0  0  0  0  0  0  0
553  405879.326<07>1:13 [347]8  44 0  0  0  0  0  0  0
554  405882.951<07>1:13 [347]6  85 0  0  0  0  0
555  405884.632<07>2:02 [43A]0
556  406307.027<07>2:05 [43D]2  10 0
557  406309.059<07>1:15 [3C7]0
558  406683.633<07>1:13 [347]8  0  6  90 33 39 35 31 39
559  406684.812<07>1:13 [347]8  41 39 30 39 35 38 37 37
560  406686.041<07>1:13 [347]8  42 0  30 0  0  0  0  0
561  406687.275<07>1:13 [347]8  43 0  0  0  0  0  0  0
562  406688.506<07>1:13 [347]8  44 0  0  0  0  0  0  0
563  406689.611<07>1:13 [347]6  85 0  0  0  0  0
564  406690.788<07>2:02 [43A]0
565  406716.164<07>1:13 [347]8  0  6  90 33 39 35 31 39
566  406717.339<07>1:13 [347]8  41 39 30 39 35 38 37 37
567  406718.578<07>1:13 [347]8  42 0  30 0  0  0  0  0
568  406719.812<07>1:13 [347]8  43 0  0  0  0  0  0  0
569  406723.444<07>1:13 [347]8  44 0  0  0  0  0  0  0
570  406724.508<07>1:13 [347]6  85 0  0  0  0  0
571  406725.685<07>2:02 [43A]0
572  406974.519<07>2:05 [43D]2  12 0                      PLC's read Ack
573  406976.527<07>1:15 [3C7]0
574  407223.836<07>1:13 [347]8  0  4  90 33 39 35 31 39
575  407225.011<07>1:13 [347]8  41 39 30 39 35 38 37 37
576  407226.275<07>1:13 [347]8  42 0  30 0  0  0  0  0
577  407227.507<07>1:13 [347]8  43 0  0  0  0  0  0  0
578  407228.741<07>1:13 [347]8  44 0  0  0  0  0  0  0
579  407229.843<07>1:13 [347]6  85 0  0  0  0  0
580  407232.14 <07>2:02 [43A]0
581  407233.908<07>1:13 [347]8  0  0  90 33 39 35 31 39   Ack acked                259.389
582  407235.11 <07>1:13 [347]8  41 39 30 39 35 38 37 37
583  407236.381<07>1:13 [347]8  42 0  30 0  0  0  0  0
584  407237.612<07>1:13 [347]8  43 0  0  0  0  0  0  0
585  407238.846<07>1:13 [347]8  44 0  0  0  0  0  0  0
586  407239.949<07>1:13 [347]6  85 0  0  0  0  0
587  407241.967<07>2:02 [43A]0
588  407265.622<07>1:13 [347]8  0  0  90 33 39 35 31 39
589  407266.837<07>1:13 [347]8  41 39 30 39 35 38 37 37
590  407268.102<07>1:13 [347]8  42 0  30 0  0  0  0  0
591  407269.333<07>1:13 [347]8  43 0  0  0  0  0  0  0
592  407270.565<07>1:13 [347]8  44 0  0  0  0  0  0  0
593  407273.487<07>1:13 [347]6  85 0  0  0  0  0
594  407274.712<07>2:02 [43A]0
595  407876.939<07>2:05 [43D]2  10 0
596  407878.948<07>1:15 [3C7]0
597  408754.248<07>2:05 [43D]2  11 0                      PLC trigger Bcode read
598  408756.273<07>1:15 [3C7]0
599  408894.496<07>1:13 [347]8  0  1  90 33 39 35 31 39
600  408895.704<07>1:13 [347]8  41 39 30 39 35 38 37 37
601  408896.976<07>1:13 [347]8  42 0  30 0  0  0  0  0
602  408898.207<07>1:13 [347]8  43 0  0  0  0  0  0  0
603  408899.439<07>1:13 [347]8  44 0  0  0  0  0  0  0
604  408900.544<07>1:13 [347]6  85 0  0  0  0  0
605  408902.128<07>2:02 [43A]0
606  408927.041<07>1:13 [347]8  0  1  90 33 39 35 31 39
607  408928.242<07>1:13 [347]8  41 39 30 39 35 38 37 37
608  408929.514<07>1:13 [347]8  42 0  30 0  0  0  0  0
609  408930.745<07>1:13 [347]8  43 0  0  0  0  0  0  0
610  408934.33 <07>1:13 [347]8  44 0  0  0  0  0  0  0
611  408935.402<07>1:13 [347]6  85 0  0  0  0  0
612  408937.698<07>2:02 [43A]0
613  409054.546<07>1:13 [347]8  0  7  90 33 39 35 31 39   Barcode read complete    300.298
614  409055.746<07>1:13 [347]8  41 39 30 39 35 38 37 37
615  409057.114<07>1:13 [347]8  42 0  30 0  0  0  0  0
616  409058.348<07>1:13 [347]8  43 0  0  0  0  0  0  0
617  409059.579<07>1:13 [347]8  44 0  0  0  0  0  0  0
618  409060.684<07>1:13 [347]6  85 0  0  0  0  0
619  409062.845<07>2:02 [43A]0
620  409086.149<07>1:13 [347]8  0  7  90 33 39 35 31 39
621  409087.373<07>1:13 [347]8  41 39 30 39 35 38 37 37
622  409088.636<07>1:13 [347]8  42 0  30 0  0  0  0  0
623  409089.868<07>1:13 [347]8  43 0  0  0  0  0  0  0
624  409091.1  <07>1:13 [347]8  44 0  0  0  0  0  0  0
625  409094.197<07>1:13 [347]6  85 0  0  0  0  0
626  409095.566<07>2:02 [43A]0
```

```
627  409688.864<07>2:05 [43D]2  10 0
628  409690.874<07>1:15 [3C7]0
629  409884.883<07>1:13 [347]8  0  6  90 33 39 35 31 39
630  409886.059<07>1:13 [347]8  41 39 30 39 35 38 37 37
631  409887.314<07>1:13 [347]8  42 0  30 0  0  0  0  0
632  409888.547<07>1:13 [347]8  43 0  0  0  0  0  0  0
633  409889.779<07>1:13 [347]8  44 0  0  0  0  0  0  0
634  409890.883<07>1:13 [347]6  85 0  0  0  0  0
635  409892.806<07>2:02 [43A]0
636  409916.852<07>1:13 [347]8  0  6  90 33 39 35 31 39
637  409918.078<07>1:13 [347]8  41 39 30 39 35 38 37 37
638  409919.34 <07>1:13 [347]8  42 0  30 0  0  0  0  0
639  409920.572<07>1:13 [347]8  43 0  0  0  0  0  0  0
640  409924.638<07>1:13 [347]8  44 0  0  0  0  0  0  0
641  409925.741<07>1:13 [347]6  85 0  0  0  0  0
642  409926.919<07>2:02 [43A]0
643  410893.232<07>2:05 [43D]2  12 0                                PLC's read Ack
644  410895.848<07>1:15 [3C7]0
645  410955.285<07>1:13 [347]8  0  0  90 33 39 35 31 39  Ack acked                  62.053
646  410956.501<07>1:13 [347]8  41 39 30 39 35 38 37 37
647  410957.775<07>1:13 [347]8  42 0  30 0  0  0  0  0
648  410959.005<07>1:13 [347]8  43 0  0  0  0  0  0  0
649  410960.239<07>1:13 [347]8  44 0  0  0  0  0  0  0
650  410961.342<07>1:13 [347]6  85 0  0  0  0  0
651  410962.909<07>2:02 [43A]0
652  410987.742<07>1:13 [347]8  0  0  90 33 39 35 31 39
653  410988.936<07>1:13 [347]8  41 39 30 39 35 38 37 37
654  410990.206<07>1:13 [347]8  42 0  30 0  0  0  0  0
655  410991.439<07>1:13 [347]8  43 0  0  0  0  0  0  0
656  410995.104<07>1:13 [347]8  44 0  0  0  0  0  0  0
657  410996.2  <07>1:13 [347]6  85 0  0  0  0  0
658  410997.792<07>2:02 [43A]0
659  411822.449<07>2:05 [43D]2  10 0
660  411826.211<07>1:15 [3C7]0
```

# Appendix D Report Sent to Developer on Developing their own Run-time Environment

This is a copy of the report that was sent to the client on what would be involved if they developed their own IEC 6499 run-time environment. (Meek, 2003a):

## Report into Requirements for Developing an IEC 61499 Run-time Environment.

### Abstract

This report gives an indication into the time and process requirements for IEC 61499 run-time development. The information in this report assumes that TCS will be developing everything in house.

### Stages Required for Development an an IEC 61499 Run-time Environment

The task required to achieve an entry level IEC 61499 function block programmable compatible product are as follows:

1) Have a Java development environment for an embedded platform.
2) Have an interface from the function block development tool to the embedded platform.
3) Develop a basic function block run-time environment, this needs to be equivalent to Rockwell Automation run-time environment to be entry level.
4) Conformance testing, this will most likely be an ongoing procedure.
5) At this stage the product should be at an entry level for a product to be commercialised.

6) Develop predictive control algorithm function blocks etc. This step might be ongoing over the life cycle of the function block programming technology.

## Developing a Java Platform

It seems that TCS are definitely going down the path of Java for IEC 61499 development. There are two paths for development of Java: one is to use a platform that already has Java implemented, the other is to make use of the KVM, which is a Java virtual machine that we have C source code for, so it is transportable to other platforms. Discussed below is what is required to use the KVM. As an example of a platform, development is discussed using the current M16 inside the CPU4. The size of the task converting the KVM software to another platform will probably be similar for other microprocessors.

### The KVM

The KVM is a Java virtual machine (interpreter) which has been written in C (so it is compilable on different platforms), for embedded devices. The KVM source code from research of Alistar Edgar's is freeware and we have the source code for this in house, however before it can be used this code needs to be ported from the PC platform to our target. The features of the KVM are listed below:

- In theory our M16 C compiler will compile the KVM.
- Optional floating point arithmetic routines, this is a requirement for TCS.
- JCC JavaCodeCompact. This is a feature that allows the Java software to be embedded into the virtual machine. For development it would be simpler to not use this feature, however for production this feature may simplify product assembly.
- JAM, Java application manager, for development of the function block run-time environment, this is not required.
- Asynchronous native function execution, whether this feature will be required for function block run-time environment development is unknown at this

stage. This feature could be either a support or a hindrance to real-time control.

- KNI support, this feature will need supporting as it is the interface to the C based DeviceNet stack.
- Java-level debugging options, this will be required by TCS as it is a tool that will assist development of Java software.

## What is Required to Port the KVM to an Embedded Platform

The KVM has two sections: platform independent code and platform dependant code. In theory the platform independant code should not require any work to port to the embedded platform, providing there are no incompatibilities with the target's C compiler. The platform dependant code will require modification. This consists of the following files:

- **machine_md.h**

Includes the defines, include files and prototypes for platform dependant subroutines.

- **main.c**

This includes the main service routine which will need developing to suit the platform. This task for development will be called by an external interface after the Java software has been downloaded to the device. In a production model this will be called by the bootstrap loader.

- **runtime_md.c**

This file contains the routines which are required for alerting the operator or software developer when there are serious errors, plus functions that are required for memory management. Analysing the functions for the PC platform, these routines are about a days work.

- **loaderFile.c**

Unlike C where a program is compiled down to hex code and executed, with Java, a file system of some type is required. This does not mean that it has to be

as complicated as that on a PC. What I am invisaging is a look up table which will have the file names and a pointer that points to the start and finish of each file. I do not see that we need to worry about handling fragmentation as on a PC, because with Java, the software will be only be downloaded when programming the device.

*Developing a file system and an interface to download the Java programs to the embedded platform an estimation of time is at least 2 weeks. To achieve this a file system needs and a communication protocol needs developing.*

- **jar.c**

This file handles the uncompression of Java library files.

*The size of the task in modifying this is unknown. From what I understand if we develop a good file system this task may not be an issue, however as the Java files are in Zip type formats porting this file could take a day or weeks.*

- **Debugging interface**.

With the present CPU4 platform, the emulation equipment for C has proved to be an asset that has saved hours of time. Using Java this emulation equipment will not be of much help. To use emulation in Java a debugging interface needs to be integrated with the Java development software on the PC. The KVM includes the debugging software, however the communication protocol needs to be developed. Development is required with the embedded device and the PC development software. At present the protocol is set up to be interfaced to the TCP/IP stack which is part of Windows. At this stage this should be converted to a RS-232 interface, and maybe later on use CIP (DeviceNet and EtherNet/IP).

*Getting the debugger going could take at least a month however from the experience of DeviceNet the time savings at a later stage will be well worth it.*

- **Other miscellaneous tasks**

I expect that there will be miscellaneous tasks of configuring C compilers, and developing a user friendly interface to make programming the device simple. I

believe that if everything is set up to be user friendly it will save development time.

- **Hardware requirements**

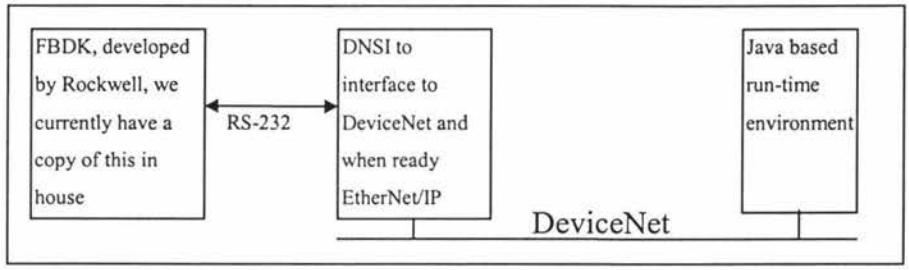
At this stage the platform to which the KVM will be ported is unknown. At present the CPU4 may be used for development purposes, however a memory upgrade will be required. A memory upgrade may not necessarily be required if all development is done on the emulator, however a memory upgrade should be performed on some of our prototype CPU4 boards. If we port to the CPU4 it should not be a big issue porting the KVM to another platform as we already know how to do it.

An estimation of time to get Java running would be about 3 months.

## Development of Tools Required to Interface an IEC 61499 Run-time Environment to the PC Platform

The plan to achieve this is unclear at this stage with regards to how everything sits with Rockwell Automation. Assuming that we are going to develop the run-time environment ourselves, this is how I invisage everything to work. Even if we use Rockwell Automation's run-time environment we will more than likely have to do our own development in this area.

**Figure D.1 Programming interface for IEC 61499 devices**




How our remote devices will interface to the FBDK software is a difficult concept to understand. When a function block program is developed it looks like a centralised

application that runs completely off the machine that the development work is done on. The user will select resources and devices that the software is going to run. With the present FBDK you can chose a local resource (ie the application will run on the development PC) or a remote resource which is a PC on the same network as the development PC. The devices and the resources in the FBDK are like the programming communication interface stacks that will communicate to the device on the network. For TCS to use the FBDK, custom TCS devices and resources can be added to the FBDK software. This will consist of writing a Java extension to the FBDK and developing a communication tool to talk to the network that the embedded device is going to use. With this set up, when the user of the FBDK presses the launch button the FBDK will download the software to the device and the application will operate.

There are a number of options for the communication networks for programming the devices. All present development uses TCP/IP over ethernet without EtherNet/IP. What I see happening in the future for TCS is the use of EtherNet/IP. At present I do not see our in house development of ethernet technologies ready for when I will be needing it. As an interim protocol DeviceNet will be used. As DeviceNet and EtherNet/IP use a common CIP component, there should be no problems converting to EtherNet/IP at a later date. The present status of our DeviceNet stack in regards to IEC 61499 is as follows: The peer-to-peer protocol has been developed but connection management capabilities are needed before it is acceptable for production use. For the time being we could use our proprietary CAN based protocol for the kiwi fruit project to get us started. The economics of developing the DeviceNet stack any further to me are unknown. Will we be using IEC 61499 function blocks commercially over DeviceNet? For starters we have a tool here which is our scanner module for John Ward Electrical.

The requirements for development of the resource and device function blocks are as follows: The FBDK supports the addition of custom resources made to interface to our devices by us writing Java code. The estimation of time required to do this task is

difficult due to lack of documentation on the FBDK. Observing remote devices at the FBDK end, we need to develop the communications interface for our RS-232 to DeviceNet (or any other) protocol converter. Assuming that by this stage of development I am up to speed with Java and from the experience gained writing the I/O function block for the FBRT (Rockwell Automation's run-time environment), I am guessing about 2 to 4 weeks worth of work.

The development of the software receiver interface at the embedded end comes under the subject of developing the software run-time environment.

## Developing the TCS Run-time

At present there are two options:

- Use the Rockwell Automation's run-time environment, this is currently under investigation and it is unknown if this will be cost effective. Also before this path is considered it must be checked against the academic requirements for Andrew Meek's Masters degree.
- Develop our own run-time environment from scratch.

The information below is presented to try and develop a potential plan to how we would go about developing our own run-time environment. The FBDK has graphical software tool in it that produces Java source code, for function blocks with the Rockwell Automation run-time environment. To simplify the development stages in the later stages of the project our run-time environment should be developed to a similar structure to the Rockwell Automation run-time environment. To achieve this from what I have observed, there is sufficient information by analysing output code from the FBDK without needing the source code of the run-time environment itself.

In terms of development, possible major milestones should consist of the following:

1) Planning of software.

2) Communications with the programming interface (FBDK).
3) The ability of being able to interface two service interface function blocks together. This application will consist of one input and one output function block interfaced with each other resulting in the input status being copied to the output.
4) Extending the above to a communications interface function block.
5) The development of basic function blocks that can perform logical operations, timers etc.
6) Development of an application that will emulate a TCS-DNSI-SICK bar code scanner interface.
7) Develop non-volatile storage for IEC 61499 applications. For all work up till this point, the software will be volatile.
8) Development of entry level function block definitions. This should include maths, logic, timers, counters, etc.
9) Development of adaptor interfaces, this is for polymorphism etc.
10) The ability of developing composite function blocks.
11) At this stage the product may be entry level for production.
12) Development of additional function blocks, such as predictive control etc. This task will be ongoing throughout the life cycle of the IEC 61499 technology.

Estimating a time for each stage is difficult, as there is no yardstick to measure against. The time estimates have been made from estimates of the problems that are likely to be encountered during the development of the IEC 61499 project.

1) A discussion with Alistair Edgar estimates a months work or planning is required before any coding is started. This planning stage should be done carefully as some of the TCS staff have not had much experience with object orientated programming.
2) Development of a programming interface will require development of the configuration application. As part of development of the configuration application, a file system and storage structure is required. This will consist of tables of function

block instances, and tables of connections (data and events). This will also consist of developing communications interfaces. *An approximation to achieve this step is 2 months however this may easily be exceeded.*

3) To achieve communication between function blocks a 'Kernal' of some type needs developing. For IEC 61499 to be appropriate for real-time automation a kernal will have to contain the following: Multi-threading, this is to allow multiple tasks within an IEC 61499 application to perform, as an IEC 61499 application will consist of multiple sub-applications. As an example, for a SICK bar code scanner interface there will be separate sub-applications to handle triggering, RS-232 communications and processing of data. Each application is built up as a network of function blocks, there needs to be a scheduling arrangement to service each of these function blocks as the networks form a circular tree structure. This makes a scheduling method complicated. Initially to get a simple application running, time slicing for multi-threading, and round robin for scheduling should get a simple application running. Talking to Alistair, Java will make some of this task simpler. *An estimate of time with Java is 2 months, however talking to Bernard Wood with his C experience has said we could be in for a big job.*
4) Extending the above to handle communications. *Estimation is one month for this task.*
5) This is to develop the basic function block definitions. From what is observed inside the code generated from the FBDK it looks like the FBDK will do most of the work required in writing this definition. To achieve this step all the function block types required to develop a SICK application will need developing. *An estimate of this is 1 to 2 months, however if a proper Kernal needs developing this will take longer.*
6) Developing a non-volatile storage system is required. This itself does not sound a big issue, however research is required here as nothing that I have seen is supported with IEC 61499 function blocks. *An estimate of time here is one month.*
7) Entry level function blocks are core function blocks, such as timers, counters, mathematical and logical operators, etc. This task consists of finding out what

function blocks are required, as a starting point this will include all the applicable function blocks that are included in the FBDK. *Most of the basic function blocks are not complex, however building up a library of small functions may take some time, an estimate of this is 2 months.*

8) Developing Adaptor interfaces, polymorphism, etc. *The time for this is unknown.*
9) Development of composite functions blocks, from observations there are two ways of doing this. The definition of a composite function block is a group of smaller function blocks. With the FBDK to create a composite function block it requires the compilation of a separate piece of source code. I am not sure if this is the way that this should be handled. I feel that in programming the device a composite function block should be broken down into smaller components and sent down. At this stage I'm not sure what the approach shall be. *An estimate of 2 months.*
10) Bug fixing and final tweaking will be required to get the product production ready.
11) Predictive control etc. *The time frame for this can not be estimated, as it is an area of research in itself.*

If TCS comes to an agreement with Rockwell Automation for their run-time environment, time can not be estimated until we know what Rockwell Automation are going to supply us. For us to use Rockwell Automation's run-time, there will no doubt be some customisation required, as Rockwell are more in the I/O market and we will be wanting functions for handling strings of data etc.

## A Calendar of Events if Developing our own Run-time Environment

The months are listed below and alongside is an indication of what should be completed at the end of them.

| | |
|---|---|
| **2003** | |
| **June** | Get 'hello world' running with Java. |
| **July** | |
| **August** | Have the Java debugger operational and the Java platform fully functioning. |
| **September** | Communications interface developed for IEC 61499 run-time environment. |
| **October** | 1) Planning of an IEC 61499 run-time environment software. |
| **November** | |
| **December** | 2) Communications with FBDK. |
| **2004** | |
| **January** | |
| **February** | 3) Simple I/O application working. |
| **March** | **Andrew will not be available for R&D as he will be Thesis Writing.** |
| **April** | **Andrew will not be available for R&D as he will be Thesis Writing.** |
| **May** | **Andrew will not be available for R&D as he will be Thesis Writing.** |
| **June** | 4) Extending 3) to include communication interfaces. |
| **July** | |
| **August** | 5/6) Have a functioning SICK application. |
| **September** | 7) Have NV storage operational. |
| **October** | |
| **November** | 8) The Library of basic function blocks is developed. |
| **December** | 9) Adaptor interfaces developed. |
| **2005** | |
| **January** | 10) Composite function blocks operational. |
| **February** | |
| **March** | 11) Product should be production ready. |
| **April** | |
| **May** | From now on develop predictive control and Holonics etc. |

The times in this chart are only estimates.

(Meek, 2003a).

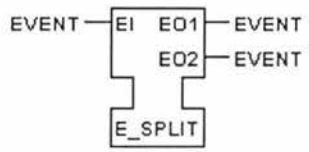
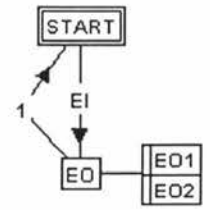
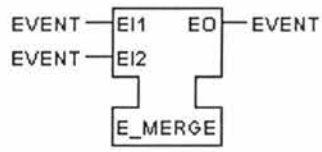
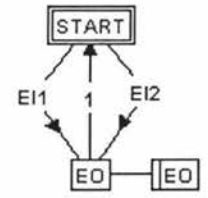
# Appendix E Function Block Library

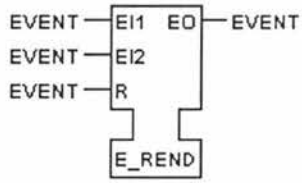
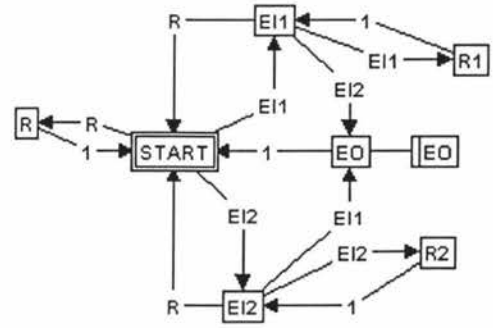
This appendix has definitions of function blocks that came with the IEC 61499 run-time environment and the function blocks that have been added to the run-time environment to satisfy the developer.

## Event Function Blocks

These function blocks handle events. The information listed below has been taken from the specification.

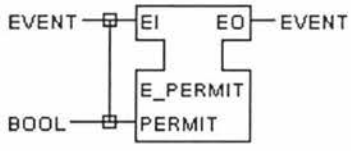
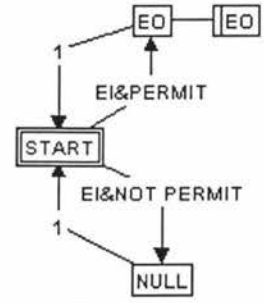
**Table E.1 Event function blocks**

<table>
<tr><th>No.</th><th colspan="2">Description</th></tr>
<tr><td colspan="2">Interface</td><td>ECC/Algorithms/Service sequences</td></tr>
<tr><td>1</td><td colspan="2">Split an event</td></tr>
<tr><td colspan="2"> 
</td><td> 
</td></tr>
<tr><td colspan="3">The occurrence of an event at EI causes the occurrence of events at EO1, EO2,...,Eon (n=2 in the above example).</td></tr>
<tr><td>2</td><td colspan="2">Merge (OR) of multiple events</td></tr>
<tr><td colspan="2"> 
</td><td> 
</td></tr>
<tr><td colspan="3">The occurrence of an event at any of the inputs EI1, EI2,...,EIn causes the occurrence of an event at EO (n=2 in the above example).</td></tr>
</table>

| 3 | Rendezvous of two events |
|---|---|
|  |  |

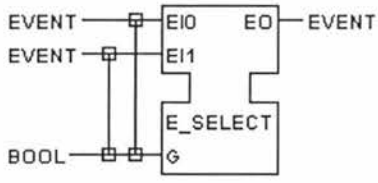
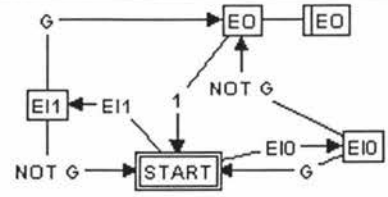




| 4 | Permissive propagation of an event |
|---|---|
|  |  |

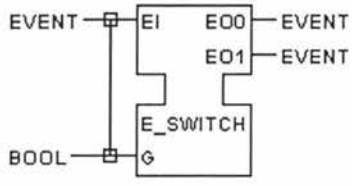
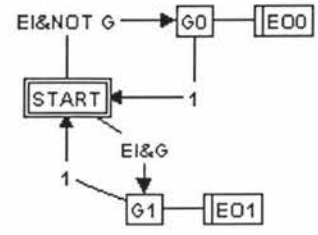


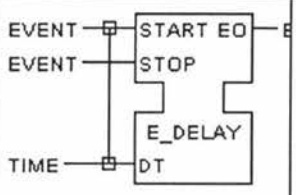


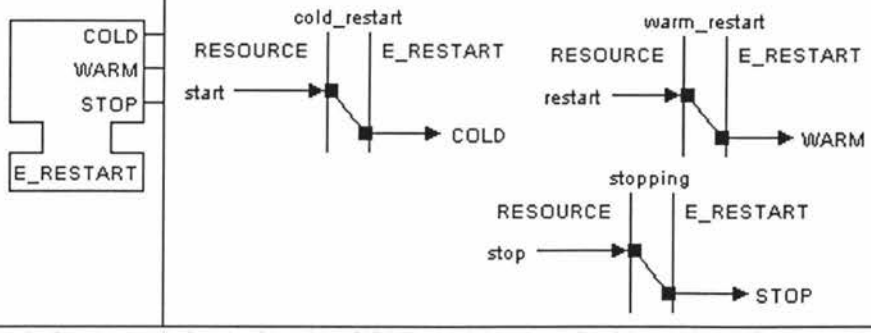
| 5 | Selection between two events |
|---|---|
|  |  |





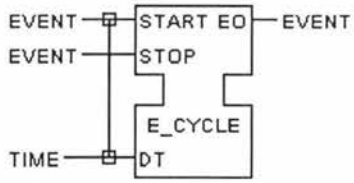
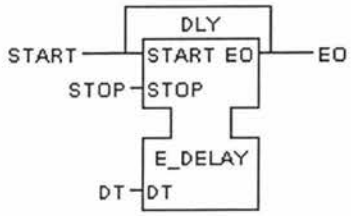
| 6 | Switching (demultiplexing) an event |
|---|---|
|  |  |

| 7 | Delayed propagation of an event |
|---|---|
|  | An event at EO is generated at a time interval DT after the occurrence of an event at the START input. The event delay is cancelled by an occurrence of an event at the STOP input. If multiple events occur at the START input before the occurrence of an event at EO, only a single event occurs at EO, at a time DT after the first event occurrence at the START input. No event delay wil be initiated if an event occurs at the START input with a value of DT which is not greater than t#0s. |

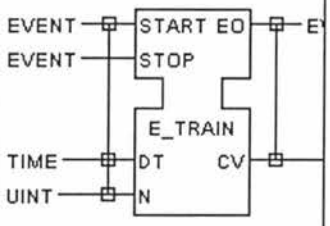
| 8 | Generation of restart events |
|---|---|
|  | |

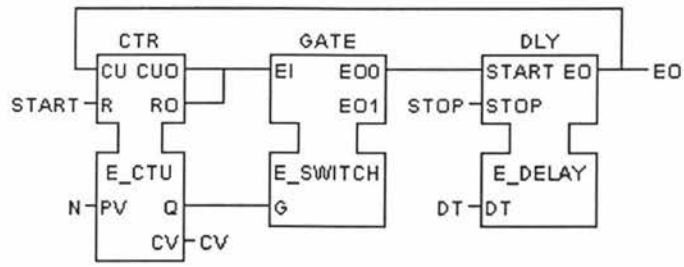
1. An event is issued at the COLD output upon "cold restart" of the associated resource.
2. An event is issued at the WARM output upon "warm restart" of the associated resource.
3. An event is issued at the STOP output (if possible) prior to "stopping" of the associated resource.

NOTE - See IEC 61131-3 for a discussion of "cold restart" and "warm restart".

| 9 | Periodic (cyclic) generation of an event |
|---|---|
|  <br>An event occurs at EO at an interval DT after the occurrence of an event at START, and at intervals of DT thereafter until the occurrence of an event at STOP. |  |

| 10 | Generation of a finite train of events |
|---|---|



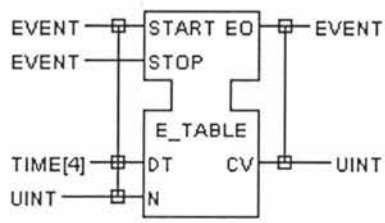



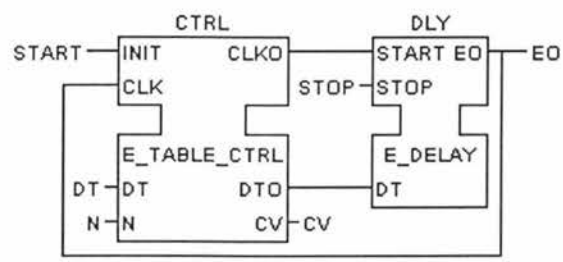

NOTE - See table entry #18 for a definition of the E_CTU type.

An event occurs at EO at an interval DT after the occurrence of an event at START, and at intervals of DT thereafter, until N occurrences have been generated or an event occurs at the STOP input.

NOTE The count CV is reset whenever an event occurs at the START interface, but the delay does not restart unless it is already stopped. This behavior maintains the inter-EO interval when restarting the count.

| 11 | Generation of a finite train of events (table driven) |
|---|---|

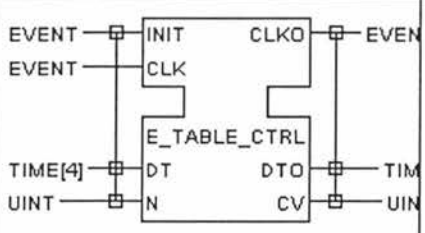


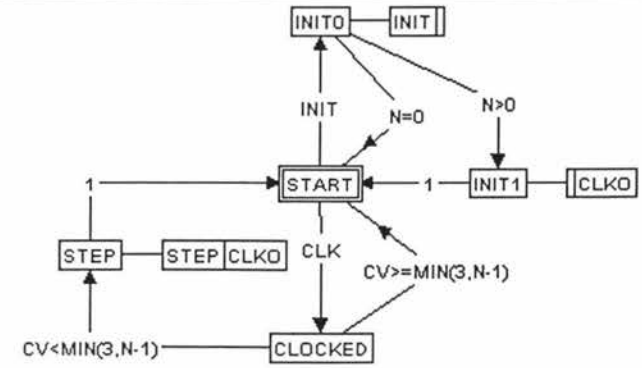




An event occurs at EO at an interval DT[0] after the occurrence of an event at EI. A second event occurs at an interval DT[1] after the first, etc., until N occurrences have been generated or an event occurs at the STOP input. The current event count is maintained at the CV output.

NOTE 1 - In this example implementation, N <= 4.

NOTE 2 - Implementation using the E_TABLE_CTRL function block type illustrated below is not a normative requirement. Equivalent functionality may be implemented by various means.

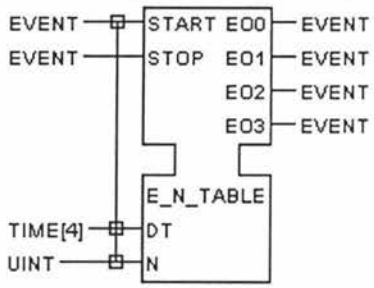


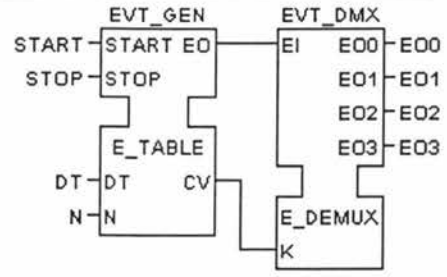




```
ALGORITHM INIT IN ST:
    CV := 0 ;
    DTO := DT[0];
END_ALGORITHM
```

```
ALGORITHM STEP IN ST:
    CV := CV+1;
    DTO := DT[CV];
END_ALGORITHM
```

| 12 | Generation of a finite train of separate events (table driven) |
|---|---|

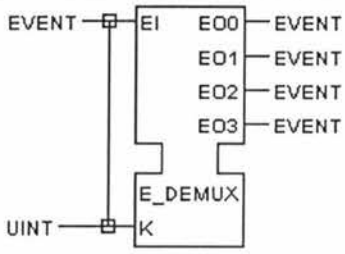


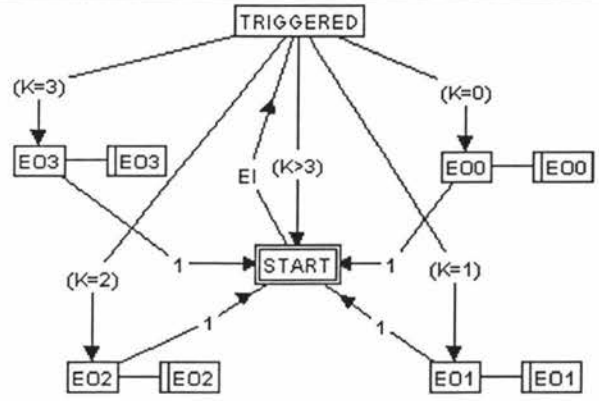




An event occurs at EO0 at an interval DT[0] after the occurrence of an event at EI. An event occurs at EO2 an interval DT[1] after the occurrence of the event at EO1, etc., until N occurrences have been generated or an event occurs at the STOP input.

NOTE 1 - In this example implementation, N <= 4.

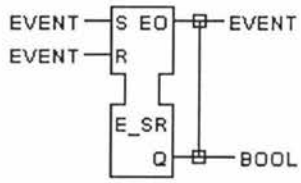
NOTE 2 - Implementation using the E_DEMUX function block type illustrated below is not a normative requirement. Equivalent functionality may be implemented by various means.

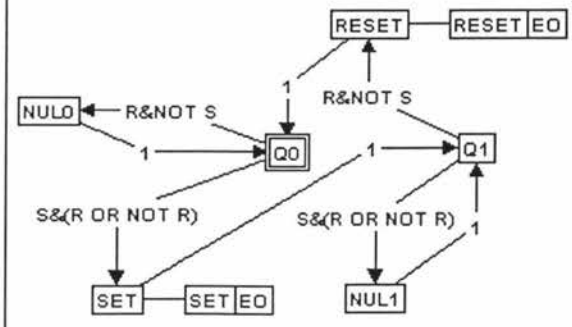



| 13 | Event-driven bistable (Set dominant) |
|---|---|

The output Q is set to 1 (TRUE) upon the occurrence of an event at the S input, and is reset to 0 (FALSE) upon the occurrence of an event at the R input. If simultaneous S and R events occur, the S input is dominant. An event is issued at the EO output when the value of Q changes.



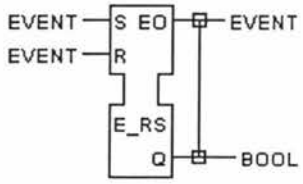



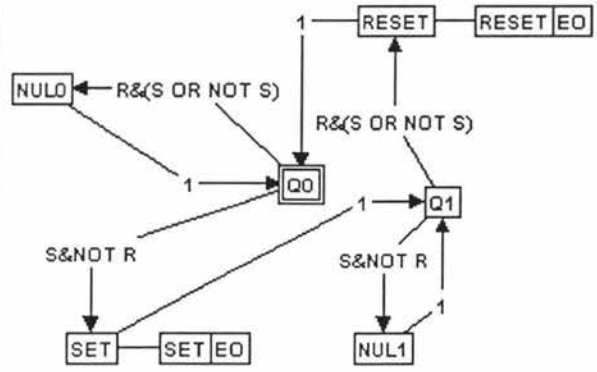

```
ALGORITHM SET IN ST :    (* Set Q
*)
    Q := TRUE ;
END_ALGORITHM
```

```
ALGORITHM RESET IN ST : (* Reset Q *)
    Q := FALSE ;
END_ALGORITHM
```

## 14 Event-driven bistable (Reset dominant)

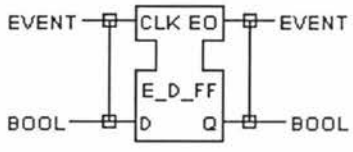
The output Q is set to 1 (TRUE) upon the occurrence of an event at the S input, and is reset to 0 (FALSE) upon the occurrence of an event at the R input. If simultaneous S and R events occur, the R input is dominant. An event is issued at the EO output when the value of Q changes.



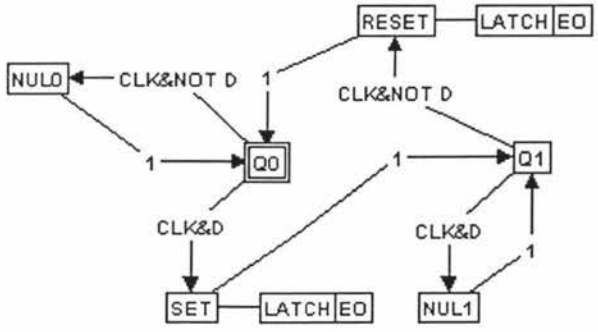




NOTE - Algorithms SET and RESET are the same as for E_SR.

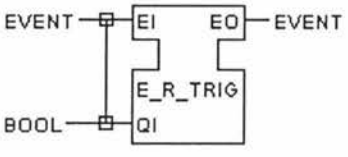
## 15 D (Data latch) bistable

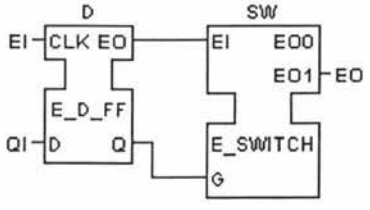



```
ALGORITHM LATCH IN ST :
  Q := D ;
END_ALGORITHM
```

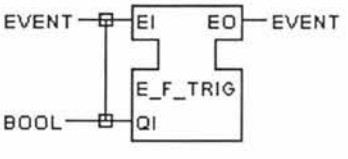


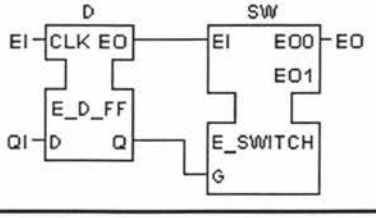

## 16 Boolean rising edge detection

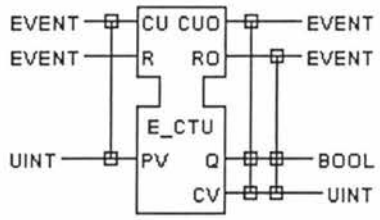
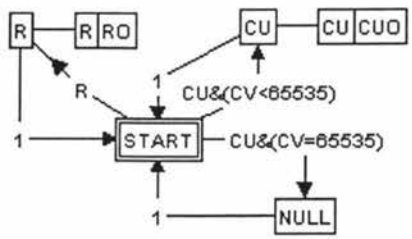



## 17 Boolean falling edge detection



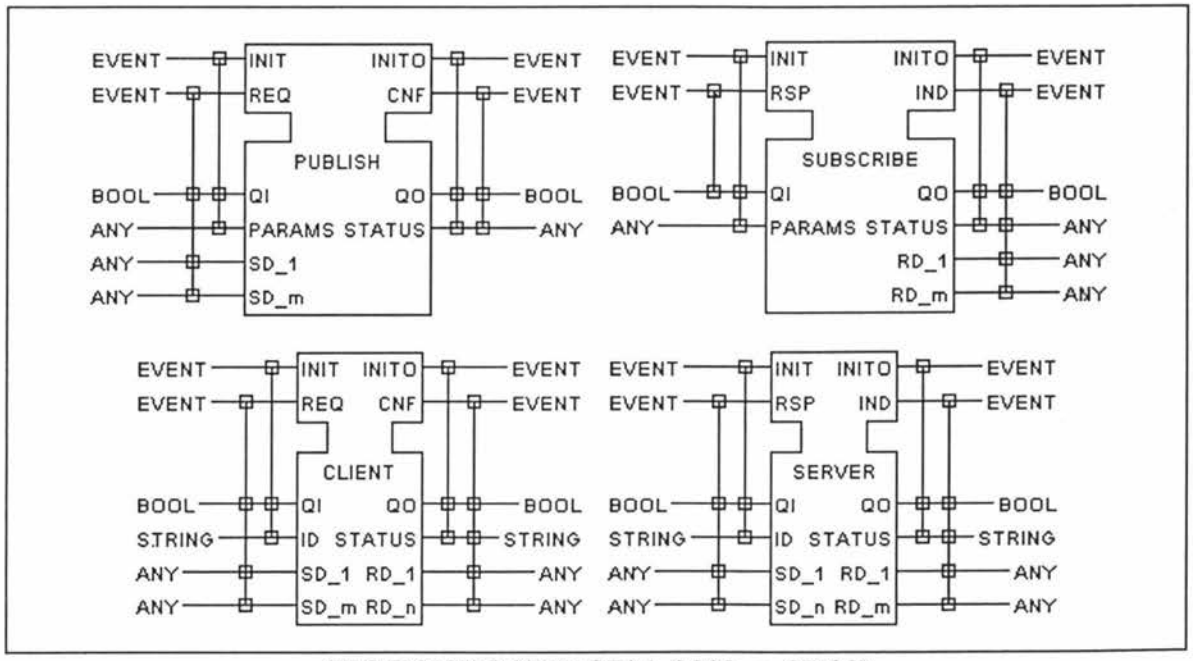
| 18 | Event-driven Up Counter |
|---|---|
|  |  |
| ALGORITHM R IN ST: (*<br>Reset *)<br>CV := 0 ;<br>Q := 0 ;<br>END_ALGORITHM | ALGORITHM CU IN ST: (* Count Up *)<br>CV := CV+1;<br>Q := (CV = PV);<br>END_ALGORITHM |

(IEC T65/WG6(PT1CD)4, 2003, p. 63-69)

## Information Exchange Function blocks

Information exchange function blocks are used for transferring data between devices and resources. The information listed below has been taken from the specification.

**Figure E.1 Information exchange function blocks**

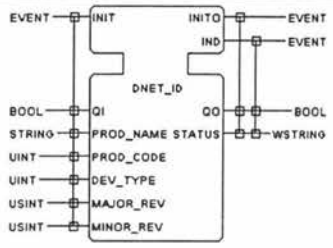
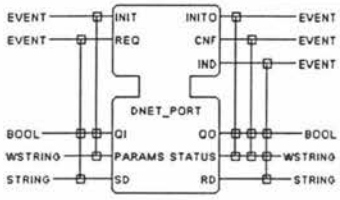
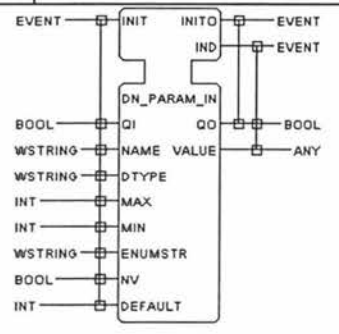


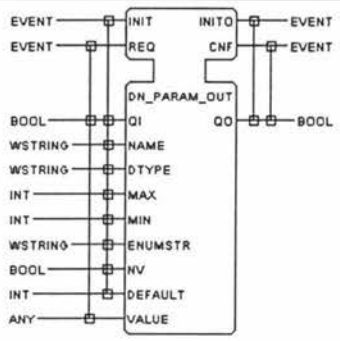

(IEC T65/WG6(PT1CD)4, 2003, p. 97-98)

## DeviceNet Specific Function Blocks

These function blocks have been added to handle the DeviceNet communication interface.

**Table E.2 DeviceNet communications function blocks**

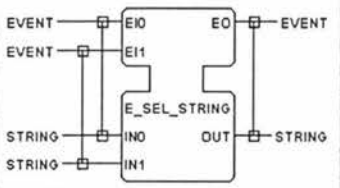
| | |
|---|---|
| **DNET_ID – Configures the DeviceNet identity object and initialises the stack.** | |
|  | QI - Enable DeviceNet<br>PROD_NAME - The name of the DeviceNet device.<br>PROD_CODE - Device's unique product code.<br>DEV_TYPE - Device type code.<br>XXX_REV - Devices revision number. |
| **DNET_PORT - Master/Slave connection interface** | |
|  | SD - Is the serial port transmit data.<br>RD - Is the serial port receive data.<br>STATUS - Is the status of the serial port.<br>PARAMS - Are the parameters for configuration, syntax is: "id=A;type=poll;consize=B;prodsize=C" where A = identifier number, B = number of bytes to consume and C = number of bytes to produce. |
| **DN_PARAM_IN, DN_PARAM_OUT - Parameter class input** | |
|  | QI - Enable parameter access.<br>NAME - The name of the parameter.<br>MAX - The maximum value of the parameter.<br>MIN - The minimum value of the parameter.<br>ENUMSTR - Comma delimited format of enumerated strings.<br>NV - Parameter non-volatile.<br>DEFAULT - Parameter default value. |

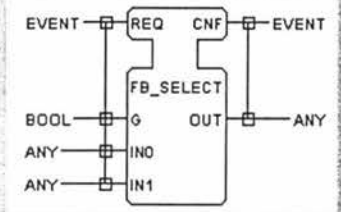
| | |
|---|---|
|  | VALUE - The value of the parameter. |

## Mathematical Function Blocks

These function blocks perform mathematical and logical operations.

**Table E.3 Mathematical operations function blocks**

| | |
|---|---|
| | **E_SEL_STRING - Event-driven STRING selection** |
| | **E_SEL_WSTRING - Event-driven WSTRING selection** |
|  | Selects an input string to transfer to the output depending on the event. |
| | **FB_SELECT - Selects two ANY data types** |
| | **FB_SEL_BOOL - Selects two BOOL data types** |
| | **FB_SEL_DINT - Selects two DINT data types** |
| | **FB_SEL_INT - Selects two INT data types** |
| | **FB_SEL_REAL - Selects two REAL data types** |
| | **FB_SEL_STRING - Selects two STRING data types** |
| | **FB_SEL_TIME - Selects two TIME data types** |
| | **FB_SEL_UINT - Selects two UINT data types** |
| | **FB_SEL_WSTRING - Selects two WSTRING data types** |

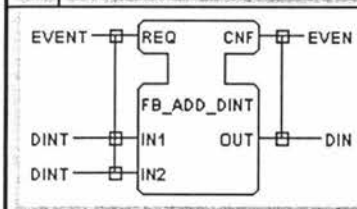



The boolean value of G is used to select the two inputs.

**FB_ADD_DINT - Addition of DINT's**

**FB_ADD_INT - Addition of INT's**

**FB_ADD_REAL - Addition of REAL's**

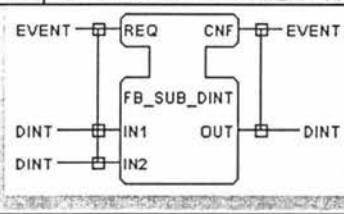



Upon a 'REQ' event IN1 and IN2 are added together and sent to the output.

**FB_SUB_DINT - Subtraction of DINT's**

**FB_SUB_INT - Subtraction of INT's**

**FB_SUB_REAL - Subtraction of REAL's**

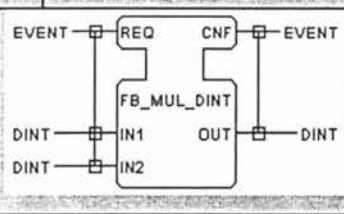



Upon a 'REQ' event IN1 and IN2 are subtracted and sent to the output.

**FB_MUL_DINT - Multiplication of DINT's**

**FB_MUL_INT - Multiplication of INT's**

**FB_MUL_REAL - Multiplication of REAL's**

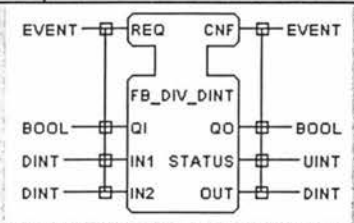



Upon a 'REQ' event IN1 and IN2 are multiplied and sent to the output.

**FB_DIV_DINT - Division of DINT's**

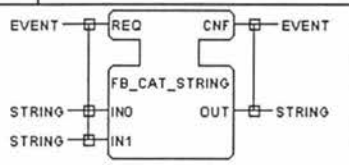
**FB_DIV_INT - Division of INT's**

**FB_DIV_REAL - Division of REAL's**




Divides IN1 by IN2, with QI enabled, the result is sent to the output. Upon successful division the QO returns positive.

**FB_CAT_STRING - Concatenates two STRINGS**

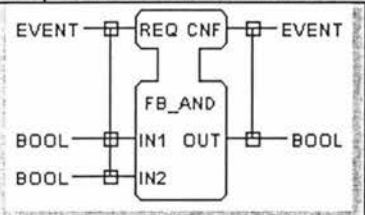



IN1 will be concatenated on the end of IN0 and sent out of OUT.

**FB_AND - Boolean AND**

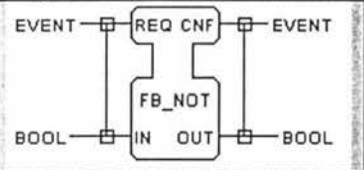
**FB_OR - Boolean OR**

**FB_XOR - Boolean XOR**



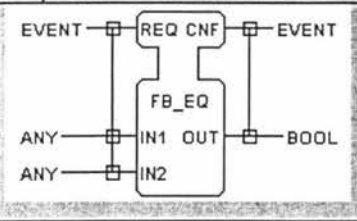

Performs a boolean AND, OR and XOR of two boolean inputs.

**FB_NOT - Boolean NOT**




Performs a boolean NOT of one boolean input.

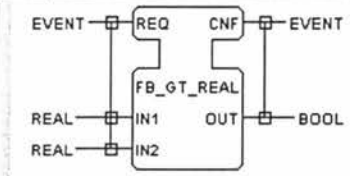
**FB_EQ - Equals comparison of two inputs**




Returns a TRUE on the boolean output if both inputs are equal to each other.

**FB_GT_REAL - Greater than comparison of REAL two inputs**

**FB_LT_REAL – Less than comparison of REAL two inputs**



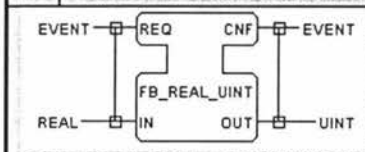

Returns TRUE on the boolean output if the inputs are:

- IN1 > IN2 for FB_GT_REAL.
- IN1 < IN2 for FB_LT_REAL.

**FB_REAL_UINT – Conversion of REALs to UINTs**

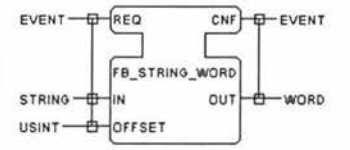
**FB_UINT_INT - Conversion of UINTs to INTs**

**FB_UINT_REAL - Conversion of UINTs to REALs**



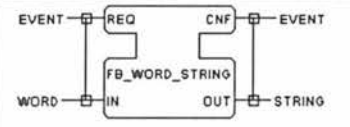

Converts data type to another data type.

**FB_STRING_WORD - Conversion of STRINGs to WORDs.**



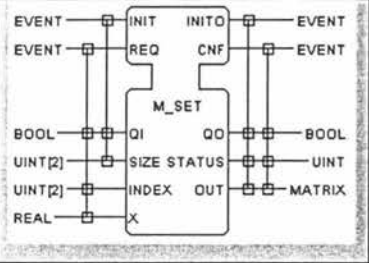

The STRING on the input starting at offset will be converted to an output WORD.

**FB_WORD_STRING - Conversion of WORDs to STRINGs**



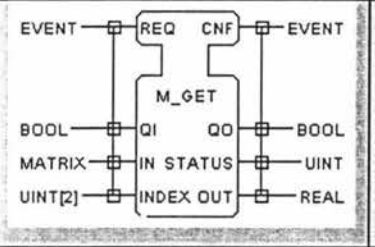

The WORD on the input is converted into a STRING.

**M_SET - Matrix set up**




Used to set up a matrix.

## M_GET - Retrieves matrix elements



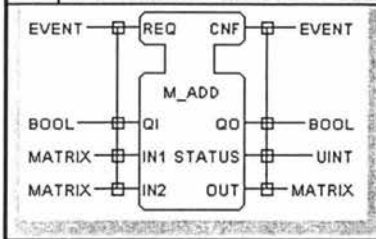

Used to get matrix elements

## M_ADD - Adds two matrices of REAL

## M_SUB - Subtracts two matrices of REAL

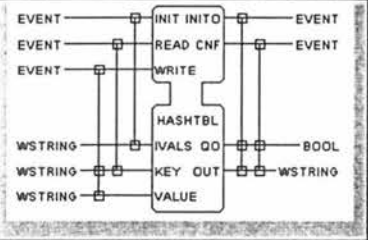
## M_MUL - Multiplies two matrices of REAL

## M_INV - Solves the matrix equation of IN1*OUT for OUT by inverting IN1

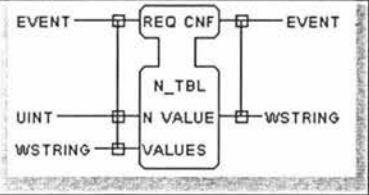



Matrix functions

## HASHTBL - Service interface for java.util.Hashtable

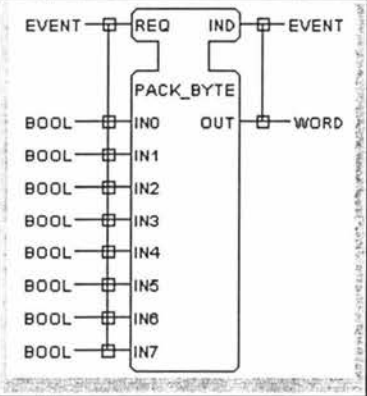



## N_TBL - Gets the Nth element form a list of strings



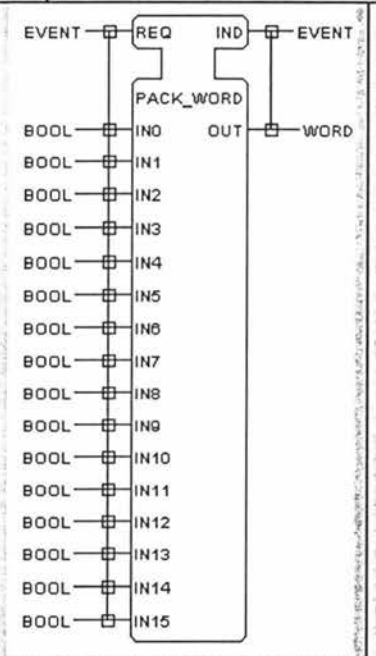

Upon a REQ it gets the Nth value of a coma-separated list of strings, which is in the VALUES input.

## PACK_BYTE - Packs a group of boolean inputs into a byte.



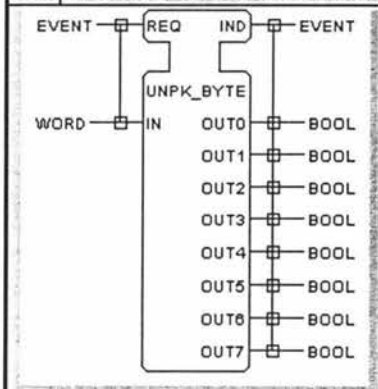

The boolean inputs are connected together to form a byte output.

## PACK_WORD - Packs a group of boolean inputs into a word.



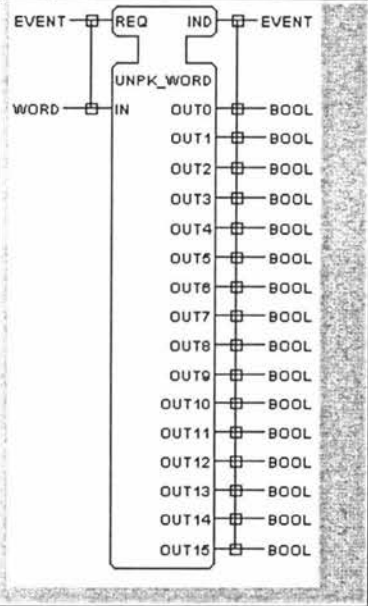

The boolean inputs are connected together to form a word output.

**UNPK_BYTE - Unpacks a BYTE to a group of BOOL outputs.**



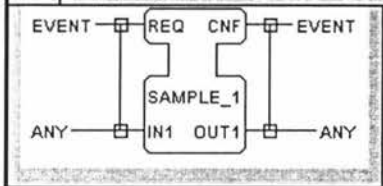

The byte input are unpacked to form a group of boolean outputs.

**UNPK_WORD - Unpacks a WORD to a group of BOOL outputs.**




The WORD input are unpacked to form a group of BOOL outputs.

**SAMPLE_X - Data variable sampling.**



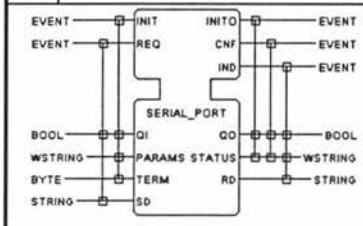
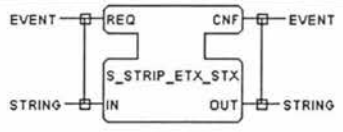

Upon a REQ event the input is transferred to the output.

(Diagrams and some explanations of that which is shaded are from FBDK, 2003)

## Serial Communication Function Blocks

These function blocks are used for serial communications.

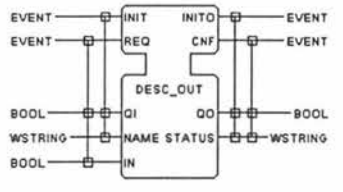
**Table E.4 Serial communications function blocks**

| | |
|---|---|
| **SERIAL_PORT - Service interface function block for the serial port** | |
|  | SD - Is the serial port transmit data.<br>RD - Is the serial port receive data.<br>TERM - Is the termination character of the received serial string.<br>STATUS - Is the status of the serial port.<br>PARAMS - Are the parameters for configuration syntax is: "comm:x;baudrate=value" |
| **S_STRIP_ETX_STX - Removes the start and end characters from a serial string** | |
|  | The STRING has the end and start characters removed from it. |

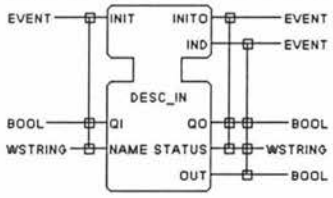
## I/O Related Function Blocks

These function blocks are related to I/O functions.

**Table E.5 I/O related function blocks**

| | |
|---|---|
| **DESC_OUT - Discrete output** | |
|  | QI – Enable<br>NAME - Name of the discrete output port.<br>IN - The output data.<br>STATUS - The status of the output port. |

## DESC_IN - Discrete input



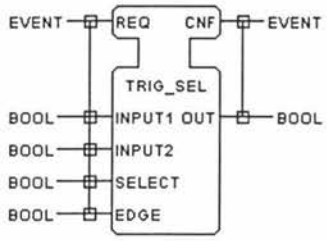

QI – Enable

NAME - Name of the discrete input port.

OUT - The input data.

STATUS - The status of the input port.

## TRIG_SEL - Selects to trigger inputs

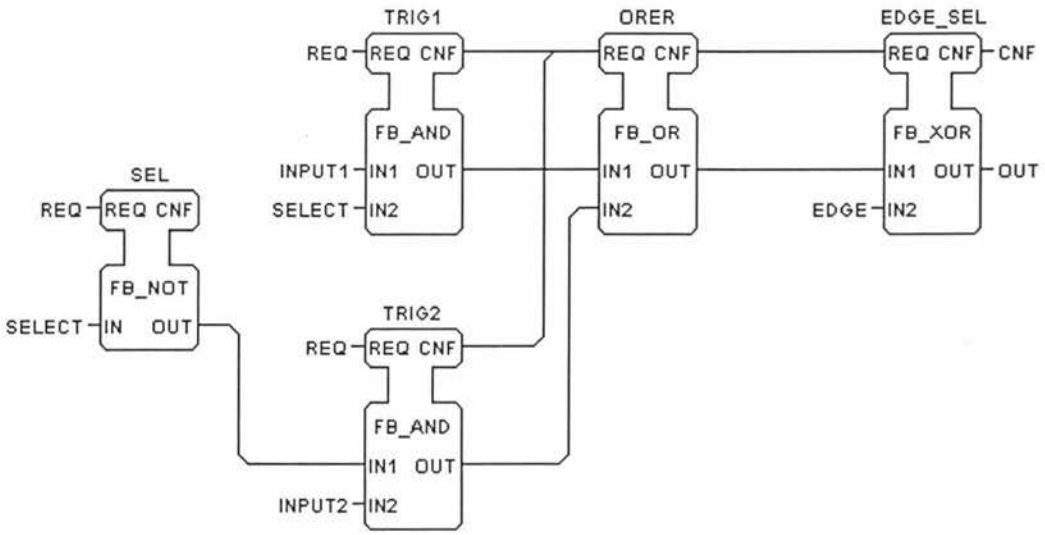



INPUT1 - Trigger input 1.

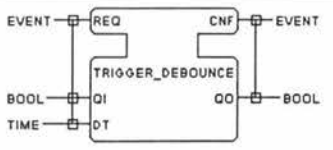
INPUT2 - Trigger input 2.

SELECT - When FALSE selects trigger input 2.

EDGE - If true negative edge triggering is used.



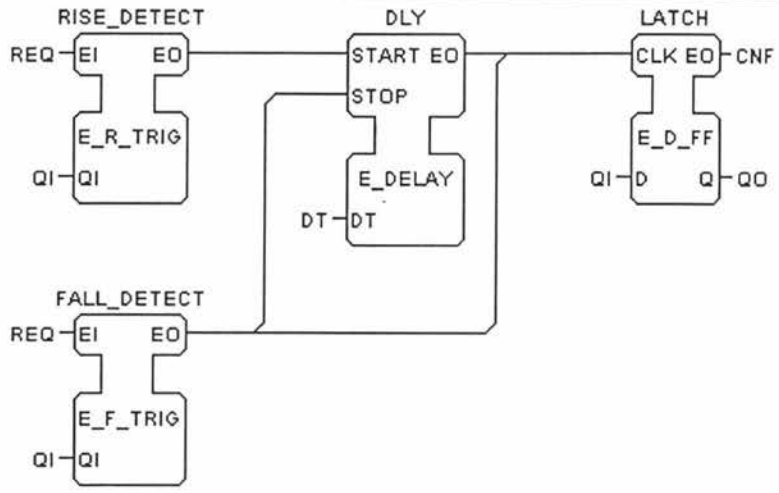

## TRIGGER_DEBOUNCE - De-bounce timer for a trigger input



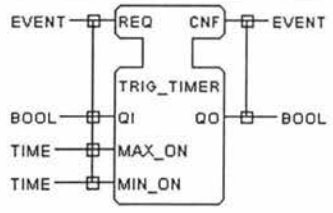

QI - Is the trigger input.

DT - Is the de-bounce time.

QO - Is the trigger output.



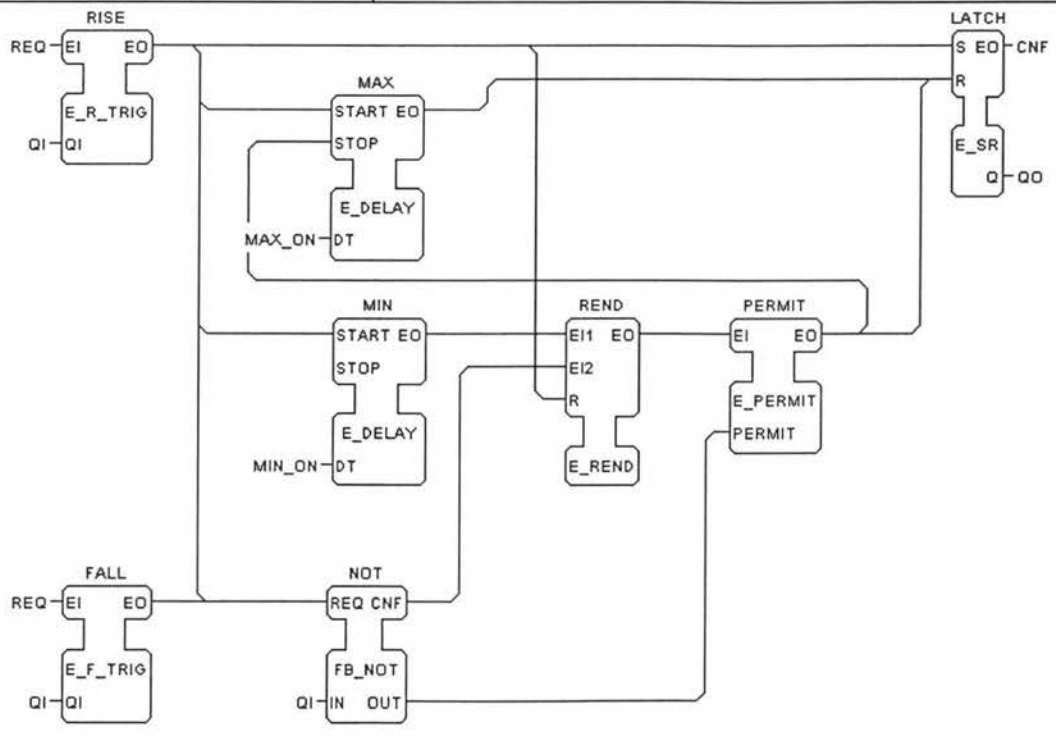

**TRIG_TIMER - Sets the maximum and minimum on times for a trigger**



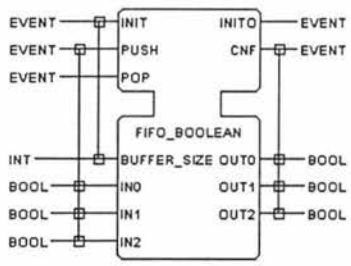

QI - The trigger input

MAX_ON - The maximum time that the QO output will remain on for regardless of QI.

MIN_ON - The minimum time that the QO output will remain on for regardless of QI.



## FIFO_BOOLEAN - Boolean first in first out buffer




INIT - initialisation.

PUSH - Pushes data into buffer.

POP - Pops data out of buffer.

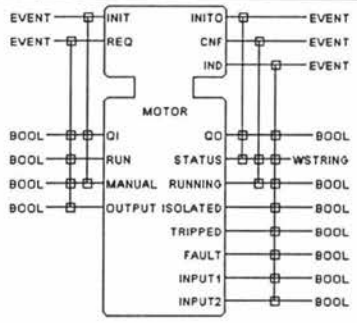
BUFFER_SIZE - The maximum size of the buffer.

CNF - Output data event.

INX - The input boolean data.

OUTX - The output boolean data.

## MOTOR - Motor control station inteface




INIT - initialisation.

REQ - request to change motor control.

QI - enable bit.

RUN - motor run control.

MANUAL - enable manual motor start control from reset button.

OUTPUT - discrete digital output control.

CNF - confirmation of request.

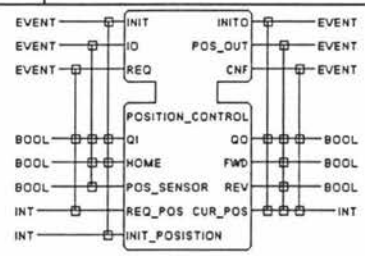
IND - indication of motor status changing.

STATUS - motor contactor and circuit breaker status.

RUNNING, ISOLATED, TRIPPED and FAULT - boolean fault outputs.

INPUTx - discrete digital inputs.

## POSITION_CONTROL - Positional control counters




IO - position control input event.

REQ - request to move to new position.

HOME - home position input.

POS_SENSOR - position sensor input.

REQ_POS - requested position.

INIT_POSISTION -requested position on initialisation.

POS_OUT - directional request.

CNF - confirmation of position.

FWD - forward position request.

REV - reverse position request.

CUR_POS - current position count.

# Appendix F DeviceNet IEC 61499 Conformance Profile

This is the draft DeviceNet IEC 61499 Conformance Profile. This document is currently under the care of Dr Jim Christensen for critiqueing before it is submitted to the standard setting body.

The document is a modified version of: "IEC 61499 Compliance Profile for Feasibility Demonstrations." (Holobloc, 2002) This is the ethernet conformance profile for IEC 61499. The following section of the profile are referenced as follows:

- **General provisions.** (Modified by author from Holobloc, 2002).
- **Portability provisions.** (Holobloc, 2002).
- **Interoperability provisions.** (Modified by author from Holobloc, 2002).
- **Configurability provisions.** (Modified by author from Holobloc, 2002).
- **Annex A – Extensions to IEC 61499.** (Holobloc, 2002).
- **Annex B – An example of remote device configuration.** (Holobloc, 2002).
- **Annex C – IEC 61499 Interface Class.** Written by author and is included in appendix G.
- **Annex D – Establishment of Dynamic I/O connections with DeviceNet.** Written by the author.

# 1. GENERAL PROVISIONS

## 1.1. Scope

This document specifies the features of IEC 61499-1 and 61499-2, according to the guidelines given in IEC 61499-4, to be implemented in order to demonstrate:

- **interoperability** of *devices* from multiple suppliers;
- **portability** of software between software tools of multiple suppliers;
- **configurability** of devices from multiple vendors by software tools of multiple suppliers; and
- **interchangeability** of devices and resources from multiple vendors.

The features to be demonstrated are illustrated in Figure 1.1.

*NOTE* - The sensor/actuator links designated #1 and #2 in Figure 1.1 may be non-interoperable. However, it is intended that feasibility demonstrations conforming to this Agreement may show the transfer of events and data from sensors on one link to actuators on another link using appropriately configured and interconnected service interface function blocks.

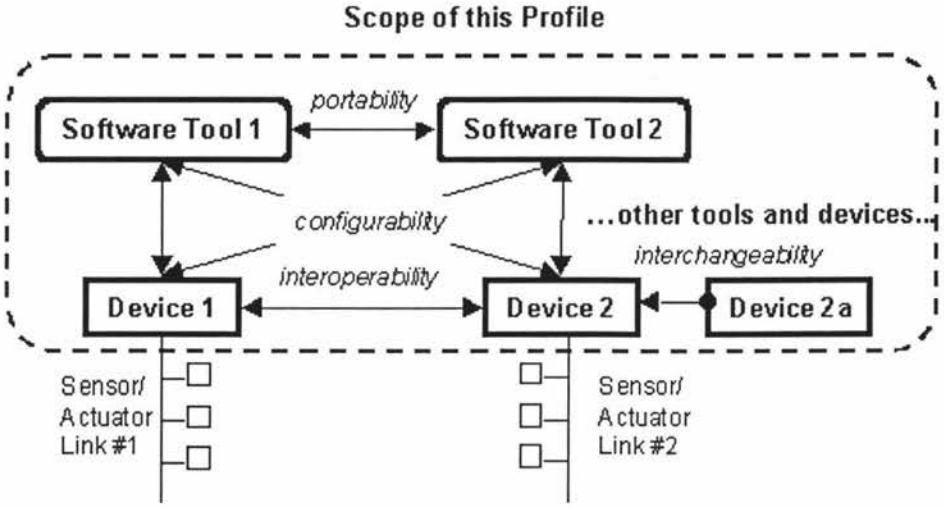



**Figure 1.1 - Scope**

## 1.2. Normative references

The following normative documents contain provisions and references to other normative documents which, through reference in this text, constitute provisions of this specification. At the time of publication, the editions indicated were valid. All normative documents are subject to revision, and parties to this agreement are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. Members of the IEC and ISO maintain registers of currently valid International Standards.

- IEC 61131-3 - Programmable controller systems, Part 3 - Programming Languages
- IEC PAS 61499-1 - Publicly Available Specification - Function blocks, Part 1 - Architecture
- IEC PAS 61499-2 - Function Blocks, Part 2 - Software tool requirements
- IEC PAS 61499-4 - Draft - Function Blocks, Part 4 - Rules for compliance profiles
- ODVA DeviceNet Specifications- Release 2.0 Errata 5
- ISO/IEC 8824, Information technology - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1).
- ISO/IEC 8825: 1990, Information technology - Open Systems Interconnection - Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).

The following document is available at www.w3c.org:

- Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation 6 October 2000

## 1.3. Definitions

The definitions given in IEC 61499-1, 61499-2 and 61499-4 and the sources referenced therein apply for the purposes of this specification.

# 2. PORTABILITY PROVISIONS

*Software tools* conforming to this specification shall meet the following portability conditions:

1. A software tool shall be capable of producing all *library elements* supported by the tool in the syntax and with the semantics defined in Annex A of IEC 61499-2.
2. A software tool shall be capable of correctly parsing and interpreting all elements supported by the tool in the syntax and with the semantics defined in the XML DTDs in Annex A of IEC 61499-2. For convenience, the following links may be used to the `LibraryElement` and `DataType` DTDs.
3. When the software tools utilize files for the exchange of library elements, the names of such files may have the form `<elementName>.xml`. Alternatively, in order to facilitate directory searching and file identification the filename suffixes shown in the following table may be used.

| Library Element | Suffix |
|---|---|
| DataType | .dtp |
| FBType | .fbt |
| AdapterType | .adp |
| ResourceType | .res |
| DeviceType | .dev |
| System | .sys |

# 3. INTEROPERABILITY PROVISIONS

*Devices* and *resources* conforming to this specification shall fulfill the following interoperability requirements of the ODVA DeviceNet Specifications Release 2.0 Errata 5.

## 3.1 Supported DeviceNet Connections

The DeviceNet Connections required for implementation for the following devices are listed in the following table:

**Table 3.1 – DeviceNet device type supported.**

| DeviceNet device type | Connection types required |
|---|---|
| Group 2 Only server | Support of Slave polled connections with the option of supporting Change of State/Cyclic and Bit-Strobe. Refer to Chapter 7 of the DeviceNet specification Volume I. |
| Group 2 server | Supports UCMM and Slave polled connections with the option of supporting Change of State/Cyclic and Bit-Strobe. Refer to Chapter 7 of the DeviceNet specification Volume I. |
| Group 2 client | Support of Master connection services for UCMM and Master polled connections with the option of supporting Change of State/Cyclic and Bit-Strobe. Refer to Chapter 7 of the DeviceNet specification Volume I. |
| Peer-to-peer client/server | Supports UCMM, and Dynamic I/O connections. Dynamic I/O connections are described in Chapter 3 of the DeviceNet specification Volume I. Specific requirements for Dynamic I/O are in annex D. |

The services of these layers shall be provided by the functional equivalent using the DeviceNet connection types, in an appropriate instance of an IEC 61499-1 Communication Service Interface Function Block (CSIFB). The encapsulated class corresponding to each generic CSIFB type is listed in Table 3.3. Implementation-specific details of CSIFB types are given in subclause 3.5.

**Table 3.2 - Equivalent DeviceNet connection types**

| Generic CSIFB type | DeviceNet connection types |
|---|---|
| PUBLISH | Dynamic I/O multicast messaging. |
| SUBSCRIBE | Dynamic I/O multicast messaging, Predefined Master/Slave Connections set, using the Slave connections. |
| CLIENT | Dynamic I/O client messaging, Predefined Master/Slave Connections set, using the Master connections. |
| SERVER | Dynamic I/O server messaging, Predefined Master/Slave Connections set, using the Slave connections. |

## 3.4 Presentation of Data

Each data element of data shall be grouped into attributes 3 and 4 of the IEC 61499 interface object. Refer to annex C.

### 3.4.1 Encoding of FBDataSequence elements

The encoding of an FBDataSequence element shall be **constructed**, and shall not include a type or length field. That is, this type shall be encoded simply as a sequence of FBData elements.

### 3.4.2 Encoding of the NULL element

The NULL element shall be encoded as a single byte containing the tag of the NULL type (16#05) as defined in ISO/IEC 8825.

### 3.4.3 Unsupported types

Since many run-time systems do not support unsigned 64-bit integer values, it is not required that the ULINT type be supported. It is not required that the data types DATE and

TIME_OF_DAY be supported, since strings representing their values can always be derived as substrings of a full string representation of the corresponding DATE_AND_TIME type.

## 3.4.4 Encoding of values of elementary data types

The values of elementary data type elements shall be transferred according to the following modified ASN.1 syntax:

```
ElementaryData ::= CHOICE{
BOOL,
FixedLengthInteger,
FixedLengthReal,
TIME,
AnyDate,
AnyString,
FixedLengthBitString,
ARRAY}
```

### 3.4.4.1 Encoding of time-related values

Values of the TIME data type shall be encoded as specified in Table F.3.2.2 of IEC 61499-1.

Values of the DATE_AND_TIME type shall be as specified by the following modified syntax and encoding rules for Annex F.3.1.1 and Table F.3.2.2 of IEC 61499-1.

DATE_AND_TIME::= [APPLICATION 15] IMPLICIT LINT -- See Table F.3.2.2.

| Data type | Contents octets | |
|---|---|---|
| | Length | Encoding rule |
| DATE_AND_TIME | 8 | (5) |
| ENCODING RULES FOR TABLE F.3.2.2<br>(5) Values of this type shall be encoded as for type LINT, representing the number of milliseconds since 1970-01-01-00:00:00.000 | | |

#### 3.4.4.2 Encoding of string elements

The length field of STRING and WSTRING elements shall be encoded as for the UINT type, i.e., a 16-bit unsigned integer. The encoded character sequence for both STRING and WSTRING values shall be such that the first character in the string is the first encoded

value, e.g., for the string "ABC" the first encoded character shall be "A" and the last shall be "C". The values of the individual character elements of the STRING and WSTRING types shall be encoded in the same way as the values of the USINT and UINT types, respectively.

### 3.4.4.3 Encoding of ARRAY elements

ARRAY elements shall be transferred according to the following modified ASN.1 syntax:

```
ARRAY ::= [APPLICATION 22] IMPLICIT SEQUENCE OF FBData
```

*NOTE* - Since ARRAY is a subclass of FBData, a multidimensional ARRAY may be encoded recursively as an ARRAY whose elements are ARRAY elements.

The encoding of ARRAY elements shall be **constructed** in the sense of ISO/IEC 8825, with the following provisions for COMPACT encoding:

1. The "length" subfield of the ARRAY element shall be encoded as a value of the UINT type without identifier or length octets, i.e., as a 16-bit unsigned integer.
   *NOTE* – The maximum length of an array is governed by the produced size of the DeviceNet connection.
2. The COMPACT encoding specified in IEC 61499-1 shall be used for the first element of the values field.
3. Subsequent elements, if any, shall be encoded using the COMPACT syntax without an identifier subfield.
4. If the specified length of the received ARRAY is less than the locally allocated space, the remaining elements of the local array shall be unaffected; if the length of the received ARRAY is greater than the locally allocated space, the remaining received elements shall be ignored.

## 3.4.5 Encoding of values of derived data types

Values of derived data types shall be transferred according to the following modified ASN.1 syntax:

```
DerivedData ::= CHOICE{
DirectlyDerivedData,
EnumeratedData,
SubrangeData,
```

```
ArrayData,
STRUCT}
```

### 3.4.5.1 Encoding of structured data type elements

The encoding of instances of structured data types shall be **constructed** in the sense of ISO/IEC 8825, with the following provisions for COMPACT encoding:

1. The "length" subfield shall not be used in the encoding of the main element.
2. Elements of the structure shall be encoded in the order given in the declaration of the structured data type, using the COMPACT encoding rules without an identifier subfield.

### 3.4.5.2 Encoding of enumerated data type elements

The encoding of values of enumerated data types shall use the COMPACT encoding rules, according to the following modified ASN.1 syntax:

```
EnumeratedData ::= [typeID] IMPLICIT USINT
```

The encoded value zero (0) shall correspond to the first enumerated value, with subsequent encoded values corresponding to the subsequent enumerated values.

*NOTE* - This limits the maximum number of enumerated values to 256.

### 3.4.5.3 Encoding of subrange data type elements

The encoding of values of subrange data types shall use the COMPACT encoding rules, according to the `IEC61499-FBDATA` syntax given in IEC 61499-1-F.3.1.1.

### 3.4.5.4 Encoding of array data type elements

Values of array data types shall be transferred according to the following modified ASN.1 syntax:

```
ArrayData ::= [typeID] IMPLICIT ARRAY
```

The encoding of values of array data types shall follow the rules given in 3.4.4.3, where the `[APPLICATION 22]` tag is replaced by the assigned tag of the array type.

## 3.5 Application layer

1. Communication service interface function blocks (CSIFBs) for unidirectional data exchange among devices shall be instances of the PUBLISH and SUBSCRIBE function block types defined in Annex F.2.1 of IEC 61499-1.
2. CSIFBs for bidirectional data exchange among devices shall be instances of the CLIENT and SERVER function blocks defined in Annex F.2.2 of IEC 61499-1.
3. CSIFB types for the exchange of defined numbers of data items shall have the following type name formats:
    - CLIENT_NI_NO or SERVER_NI_NO, where NI is the number of data items to be transmitted and NO is the number of data items to be received.
    - CLIENT_N or SERVER_N, where N is the number of data items to be both transmitted and received.
    - PUBLISH_N or SUBSCRIBE_N, where N is the number of data items to be transmitted or received, respectively.
4. A value of zero (0) for N, NI or NO in the above formats indicates that a corresponding NULL data item, encoded as specified in subclause 3.4.2, is to be transmitted or received.
    - *EXAMPLE* - A SERVER function block type with two data inputs SD_1 and SD_2 and a single data output RD_1 would have the type name SERVER_2_1. A corresponding CLIENT function block type would have a single data input SD_1 and two data outputs RD_1 and RD_2, and would have the type name CLIENT_1_2.
5. The PARAMS input of the standard CSIFBs shall be renamed ID and shall be of type WSTRING (this substitution is allowed by Table 3.1.1 of IEC 61499-1). The contents of this string shall correspond to the following:
    - mac=X. The MAC ID of the device that the device will be communicating to. This is only required for Dynamic I/O SERVER and SUBSCRIBE CSIFBs and Master/Slave connections with CLIENT and SUBSCRIBE CSIFBs.

- epr=X, where X is the expected packet rate for that connection. This only applies to CLIENT and PUBLISH CSIFBs.
- id=X, where X is the identifier of the IEC 61499 interface object used for this communication connection.
- prodsize=X. This is the packet size of the produced data. This only applies to CLIENT and PUBLISH SCIFBs.
- consize=X. This is the packet size of the consumed data. This only applies to CLIENT and PUBLISH SCIFBs.

6. The STATUS output of the standard CSIFBs shall be of type WSTRING. The values of this string and their corresponding semantics shall be as shown in Table 3.5.

**Table 3.5 - STATUS output values**

| Value | Corresponds to | Semantics |
|---|---|---|
| "OK" | INITO+, CNF+, IND+ | Valid operation |
| "INVALID_ID" | INITO- | Invalid ID input |
| "TERMINATED" | INITO- | Service termination |
| "INVALID_OBJECT" | CNF-, IND- | Invalid object type received or requested to be sent |
| "DATA_TYPE_ERROR" | CNF-, IND- | Received object type does not match actual output type |
| "INHIBITED" | CNF- | Caused by REQ- |
| "NO_CONNECTION" | CNF- | Unable to communicate to serve REQ+ |
| Other values | INITO-, CNF-, IND- | Socket service errors |

# 4. CONFIGURABILITY PROVISIONS

*Software tools* and *devices* conforming to this specification shall satisfy the following requirements for *configurability*.

## 4.1 Software tools

*Software tools* shall be capable of utilising the management capabilities of *devices* that are *configured* according to the functional equivalent of Figure 4.3-1.

*EXAMPLE* – The parameters will include configuration information, typically to access a remote device with a MAC identifier of 2 and an identifier of 1. The parameter or id string will be "mac=2;id=1".

## 4.2 Device management services

The device management services to be implemented are provided by the functional equivalent of an instance of the DEV_MGR type shown in Figure 4.2-1. The types and semantics of the inputs and outputs of this type are identical to the correspondingly named inputs and outputs of the MANAGER type defined in subclause 3.3.2 of IEC 61499-1, with the following differences:

1. The DST input designates the destination of the RQST input as follows:
    - A value of "" (the empty string) designates the device;
    - A value containing an IEC 61131-3 identifier designates a resource within the device;
    - A value containing a sequence of IEC 61131-3 identifiers separate by periods (the "." character) indicates a resource in a containment hierarchy of resources, with the leftmost identifier corresponding to the outermost resource and the rightmost identifier corresponding to the innermost resource.
    - *EXAMPLE 1* - A DST value of "RES1" indicates that the RQST input is destined for a resource named RES1 contained in the managed device.

- *EXAMPLE 2* - A DST value of "MOTOR1.WINDING2" indicates that the RQST input is destined for a resource named WINDING2 contained in a resource named MOTOR1 which is contained in the managed device.

2. The RQST input and RESP outputs are encoded according to the Request and Response elements, respectively of the XML DTD given in subclause 4.4. The semantics of these elements shall be as defined in subclause 4.5.
3. As illustrated in Figure 4.2-1, a REQ+ primitive input always results in a CNF+ primitive output, since the actual result including failure conditions is encoded in the RESP output. Similarly, a REQ- input always results in a CNF- output, since no management operation is attempted in this case. In particular, this means that, in an instance of the DM_KRNL function block type shown in Figure 4.3-2, an IND- primitive from the communication service interface will neither cause a management operation to be performed, nor will a response message be generated.

The sequences of service primitives for device management shall be as shown in Figure 4.2-1. The object denoted manager in these service sequences is an instance of class FBManager described in Annex C.2 of IEC 61499-1. This is the manager of the *device* or a contained *resource* depending on the value of the DST input.

## 4.3 Devices

Management of *devices* shall be accomplished by the functional equivalent of the configuration shown in Figure 4.3-1 in each device type. The DM_KRNL function block type used in this configuration is shown in Figure 4.3-2.

Suppliers of devices shall provide the equivalent of the value of the MGR_ID input of their instance of the RMT_DEV type shown in Figure 4.3-1, that is, the value of the IEC 61499 interface object element. This value will typically be "id=1". This value may be defined as part of a library element file for the device type, or may be configured through some means beyond the scope of this specification, for instance via a local serial port or configuration file.

## 4.4 FBMGT Document Type Definition (DTD)

The Request and Response elements defined in the FBMGT DTD represent the XML syntax for the RQST input and RESP output, respectively, of the DM_KRNL function block type. Explanations of the elements of this DTD, and (where applicable) references to the formal syntax for their attributes, are given in Table 4.4-2. Allowable combinations of elements, and constraints on their usage, are as given in Table 5.2 of IEC 61499-1 for the various device classes.

An example of the use of these elements is given in Annex A.

*NOTE* - To provide compact messaging, the prolog and Misc* components used in the XML document production are not used in FBMGT messages since these components are implicit in the management context; thus, only the Request or Response element is transmitted as the management message.

**Table 4.4-2 - FBMGT DTD Elements**

| Element >>Attributes | Textual Syntax (IEC 61499-1, Annex B) | Explanation |
|---|---|---|
| Request | -- | An XML-encoded management request. |
| >>ID | -- | A unique identifier for the Request/Response transaction. |
| >>Action | -- | The requested operation to be performed. See IEC 61499-1, Table 3.3.2-1. |
| Response | -- | An XML-encoded management response. |
| >>ID | -- | A unique identifier for the Request/Response transaction. |
| >>Reason | -- | A reason for failure to perform a requested action. If absent, the action has been successfully performed. See Table 4.5-1. |

| | | |
|---|---|---|
| NameList | identifier {',' identifier} | A list of FB type or data type names. |
| FBList | fb_instance_reference {',' fb_instance_reference} | |
| FBStatus | See IEC 61499-1, Figure 3.3.3-1. | |
| ByteData | Implementation-dependent data, typically encoded in hexadecimal format. | |
| VersionInfo | The currently loaded or to be loaded version of a FB type or data type. | |
| >>Organization | The organization supplying this library element | |
| >>Date | The release date of this version in YYYY-MM-DD format | |
| FB | A function block or resource instance as defined in IEC 61499-1. | |
| >>Name | fb_instance_reference | The name of the FB or resource instance |
| >>Type | fb_type_name | The FB or resource type name |
| Connection | An *event connection, data connection* or *adapter connection.* | |
| >>Source | See NOTE 1. | |
| >>Destination | See NOTE 1. | |
| VarDeclaration | A declaration of a *variable.* | |
| >>Name | input_variable_name \| output_variable_name | See NOTE 2. |
| >>Type | data_type_name | |
| >>ArraySize | See NOTE 3. | |
| >>InitialValue | See NOTE 4. | |
| FBType | An FBTypeDeclaration as described in IEC 61499-1-C.1.1. | |

| | | |
|---|---|---|
| >>Name | fb_type_name | |
| Event | A declaration of an *event interface.* | |
| >>Name | event_input_name \| event_output_name | See NOTE 5. |
| >>Type | event_type | |
| >>With | (input_variable_name {',' input_variable_name}) \| (output_variable_name {',' output_variable_name}) | See NOTE 6. |
| AdapterDeclaration | A declaration of a *plug* or *socket interface* of a *function block type.* | |
| >>Name | plug_name \| socket_name | See NOTE 7. |
| >>Type | adapter_type_name | |
| AdapterType | A declaration of an *adapter interface type* per IEC 61499-1-2.5 | |
| Name | adapter_type_name | |
| DataType | | See IEC 61131-3-2.3. |
| >>Name | data_type_name | |
| ASN1Tag | ASN.1 tag per ISO/IEC 8824-5.8. | |
| >>Class | ASN.1 tag class per ISO/IEC 8824-5.8. | |
| >>Number | ASN.1 tag number per ISO/IEC 8824-5.8. | |
| DirectlyDerivedType | See IEC 61131-3 Tables 12 and 14, #1 | |
| >>BaseType | elementary_type_name | |
| >>InitialValue | constant | |
| EnumeratedType | Same as NameList | A comma-separated list of enumerated values. |
| >>InitialValue | identifier | If present, shall be one of the list elements. |

| | | | |
|---|---|---|---|
| SubrangeType | -- | See IEC 61131-3 Tables 12 and 14, #3 | |
| >>BaseType | integer_type_name | | |
| >>InitialValue | signed_integer | | |
| Subrange | See IEC 61131-3 Tables 12 and 14, #3 | | |
| >>LowerLimit | signed_integer | | |
| >>UpperLimit | signed_integer | | |
| ArrayType | See IEC 61131-3 Tables 12 and 14, #4 | | |
| >>BaseType | non_generic_type_name | | |
| >>InitialValues | array_initialization | | |
| StructuredType | See IEC 61131-3 Tables 12, #5 and 14, #5 and #6 | | |
| ArrayVarDeclaration | See IEC 61131-3-2.3.3. | | |
| >>Name | structure_element_name | | |
| >>Type | array_type_name | | |
| >>InitialValues | array_initialization | | |
| SubrangeVarDeclaration | See IEC 61131-3-2.3.3. | | |
| >>Name | structure_element_name | | |
| >>Type | integer_type_name | | |
| >>InitialValue | signed_integer | | |

*NOTE 1* - Depending on the context, the syntax of a `Source` or `Destination` element should correspond to the syntax of the respective element in one of the productions `connection_end_point` or `accessed_data` given in Annex B.5 of IEC 61499-1.

*NOTE 2* - The productions `input_variable_name` and `output_variable_name` apply when the associated `VarDeclaration` element is part of an `InputVars` or `OutputVars` element, respectively.

*NOTE 3*- The syntax of this element when present shall be equivalent to the syntactic expression `(subrange {',' subrange}) | integer {',' integer}` where the non-terminals `subrange` and `integer` are as defined in Annex B of IEC 61131-3. Each term of the second form is equivalent to the

subrange `0..n-1`, where `n` is the value of the corresponding `integer` syntactic element. If this element is missing, the variable is not an array.

*NOTE 4* - The syntax of this element is the syntax for initialization of the corresponding variable type as defined in Annex B.1.4.3 of IEC 61131-3.

*NOTE 5* – The terms `event_input_name` and `event_output_name` apply when the `Event` element is part of an `EventInputs` or `EventOutputs` element, respectively.

*NOTE 6* – The expressions `(input_variable_name {',' input_variable_name})` and `(output_variable_name {',' output_variable_name})` apply when the `Event` element is part of an `EventInputs` element or an `EventOutputs` element, respectively.

*NOTE 7* - The terms `plug_name` and `socket_name` apply when the associated `AdapterDeclaration` element is part of a `Plugs` or `Sockets` element, respectively.

## 4.5 `Request/Response` semantics

The following rules shall apply to the use of the `Request` and `Response` elements defined in subclause 4.4 in the `normal_request` service sequence shown in Figure 4.2-1.

7. The `ID` attribute of the `Response` element shall be identical to the `ID` attribute of the `Request` element to which the `Response` element refers.
8. The absence of a `Reason` attribute in a `Response` element shall be used to indicate normal completion of the requested operation.
9. The use of sub-elements in `Request` and `Response` elements, and the meaning of possible `Reason` attributes of the `Response` element when the requested operation fails, shall be as defined in Tables 4.5-1 and 4.5-2.

**Table 4.5-1 -Request elements and Response Reason codes**

| Request | | Reason code |
|---|---|---|
| Action | Sub-element | |
| Any | Any | NOT_READY: The manager is not in a state that enables it to process the request. |
| | | UNSUPPORTED_CMD: The requested operation is not supported by the manager. |
| | | INVALID_OBJECT: Invalid sub-element or attribute syntax not covered by other, more specific Reason codes |
| | | INVALID_OPERATION: The specified action is not a valid operation on the specified sub-element. |
| | | OVERFLOW: A previous transaction is still pending |
| CREATE | FB | UNSUPPORTED_TYPE: The requested FB type is not known to the manager. |
| | | INVALID_OPERATION: The requested FB or resource cannot be created in its containing resource or device. |
| | | INVALID_STATE: An FB instance already exists with the specified name. |
| | Connection | NO_SUCH_OBJECT: One or both of the connection end points cannot be found. |
| | | INVALID_STATE: The specified connection already exists. |
| | FBType<br>AdapterType<br>DataType | UNSUPPORTED_TYPE: A type does not exist for a variable or adapter sub-element. |
| | | INVALID_STATE: A library element type already exists with the specified name. |

| | | |
|---|---|---|
| DELETE | FB | NO_SUCH_OBJECT: No FB instance of the specified type can be found with the specified instance name. |
| | | INVALID_STATE: The FB instance is not in the STOPPED or KILLED state. |
| | Connection | NO_SUCH_OBJECT: One or both of the connection end points cannot be found. |
| | FBType AdapterType DataType | UNSUPPORTED_TYPE: A library element of the specified type does not exist with the given type name. |
| | | INVALID_STATE: At least one instance of the specified type still exists. |
| | | INVALID_OPERATION: The specified type is undeletable. |
| START STOP KILL | FB | NO_SUCH_OBJECT: No FB instance of the specified type can be found with the specified instance name. |
| | | INVALID_STATE: The FB instance is not in a state from which the specified operation can be performed. |
| READ | Connection (Source=Location, Destination =null) | NO_SUCH_OBJECT: The specified Source location cannot be found. |
| | | See NOTE 1. |
| WRITE | Connection (Source=Data , Destination =Location) | NO_SUCH_OBJECT: The specified Destination location cannot be found. |
| | | INVALID_OBJECT: The format of the Source attribute is not correct for data to be written to the Destination location. |
| | | See NOTE 2. |

- *NOTE 1* - A WRITE **Request** contains a Connection sub-element with its Source attribute encoded according to the data_element production defined in Annex B.5 of IEC 61499-1, and with its Destination attribute encoded

according to the connection_end_point production defined in Annex B.5 of IEC 61499-1.

- *NOTE 2* - A normal **Response** to a READ **Request** will contain a Connection sub-element with the Source attribute encoded according to the data_element production defined in Annex B.5 of IEC 61499-1.

**Table 4.5-2 - QUERY Request and Response elements**

| **Request** sub-element | Normal **Response** sub-element | Abnormal **Response** Reason codes |
|---|---|---|
| **FB** (Name <> "*") | **FBStatus** | **NO_SUCH_OBJECT:** No FB instance of the specified type can be found with the specified instance name. |
| **FB** (Name = "*") | **FBList:** A list of all instances of the specified FB type. | **NO_SUCH_OBJECT:** No instances exist of the specified FB type. |
| **Connection** (Destination="*") | **EndpointList:** A list of the destinations of all connections originating at the specified source. | **INVALID_OBJECT:** The source specification is not a hierarchical name. |
| **Connection** (Source = "*") | **Connection+:** A list of the sources of all connections terminating at the specified destination. | **INVALID_OBJECT:** The destination specification is not a hierarchical name. |
| **FBType \| DataType \| AdapterType** (No sub-elements, Name <> "*") | **FBType \| DataType \| AdapterType:** The declaration of the library type with the specified name. | **UNSUPPORTED_TYPE:** The requested library type is not known to the manager. |

| | | |
|---|---|---|
| **FBType \| DataType \| AdapterType** (No sub-elements, Name = "*") | **NameList:** A list of names of all library elements of the specified type. | **UNSUPPORTED_TYPE:** There are no library elements of the specified type. |

# Annex A (informative) Extensions to IEC 61499

Table A identifies those features specified in this Compliance Profile which are not specified in the various Parts of IEC 61499, and their mapping to the corresponding elements specified in those Parts, respectively.

**Table A Extensions to IEC 61499-1 and -2**

| Feature | See | Mapping | IEC 61499 element |
|---|---|---|---|
| COMPACT encoding | 3.4 | Add to | 61499-1-F.3.2.2 |
| CSIFB naming conventions | 3.5 | Add to | 61499-1-F.2 |
| CSIFB ID input | 3.5 | Allowed usage | 61499-1 Table 3.1.1 |
| CSIFB STATUS output values | Table 3.5 | Allowed usage | 61499-1 Table 3.1.1 |
| Software tool configurability requirements | 4.1 | Add to | 61499-2-2.8 |
| DEV_MGR type | 4.2 | Replace | 61499-1-3.3.2 |
| Device management | 4.3 | Replace | 61499-1-Annex G |
| FBMGT DTD | 4.4 | Replace | 61499-1-F.3.1.2 |
| Request/Response semantics | 4.5 | Replace | 61499-1-3.3.2 |
| Interchangeability provisions | 5 | Add to | 61499-4 Table 1 |

# Annex B (informative) An example of remote device configuration

The Request/Response transactions shown below illustrate the configuration of the resource named RES1 in the remote device identified as DEV1 in the RMT_DEV_TEST example. The actual DST input to the corresponding DEV_MGR block in DEV1 is given by deleting the "DEV1" string (and full stop character '.' if any) from the indicated destination; for example, the DST string of request #85 is empty and the DST string of request #86 is "RES1".

```
DEV1: <Request ID="85" Action="CREATE" >
  <FB Name="RES1" Type="EMB_RES" />
</Request>

DEV1: <Response ID="85" />

DEV1.RES1: <Request ID="86" Action="CREATE" >
  <FB Name="FF" Type="E_SR" />
</Request>

DEV1.RES1: <Response ID="86" />

DEV1.RES1: <Request ID="87" Action="CREATE" >
  <FB Name="SB" Type="SUBSCRIBE_2" />
</Request>

DEV1.RES1: <Response ID="87" />

DEV1.RES1: <Request ID="88" Action="CREATE" >
  <FB Name="AD" Type="FB_ADD_REAL" />
</Request>

DEV1.RES1: <Response ID="88" />

DEV1.RES1: <Request ID="89" Action="CREATE" >
  <FB Name="PUB" Type="PUBLISH_1" />
</Request>

DEV1.RES1: <Response ID="89" />

DEV1.RES1: <Request ID="90" Action="CREATE" >
  <Connection Source="START.COLD" Destination="FF.S" />
</Request>

DEV1.RES1: <Response ID="90" />

DEV1.RES1: <Request ID="91" Action="CREATE" >
  <Connection Source="START.WARM" Destination="FF.S" />
</Request>
```

```
DEV1.RES1: <Response ID="91" />

DEV1.RES1: <Request ID="92" Action="CREATE" >
  <Connection Source="START.STOP" Destination="FF.R" />
</Request>

DEV1.RES1: <Response ID="92" />

DEV1.RES1: <Request ID="93" Action="CREATE" >
  <Connection Source="FF.EO" Destination="SB.INIT" />
</Request>

DEV1.RES1: <Response ID="93" />

DEV1.RES1: <Request ID="94" Action="CREATE" >
  <Connection Source="SB.INITO" Destination="PUB.INIT" />
</Request>

DEV1.RES1: <Response ID="94" />

DEV1.RES1: <Request ID="95" Action="CREATE" >
  <Connection Source="SB.IND" Destination="AD.REQ" />
</Request>

DEV1.RES1: <Response ID="95" />

DEV1.RES1: <Request ID="96" Action="CREATE" >
  <Connection Source="AD.CNF" Destination="PUB.REQ" />
</Request>

DEV1.RES1: <Response ID="96" />

DEV1.RES1: <Request ID="97" Action="CREATE" >
  <Connection Source="SB.RD_1" Destination="AD.IN1" />
</Request>

DEV1.RES1: <Response ID="97" />

DEV1.RES1: <Request ID="98" Action="CREATE" >
  <Connection Source="SB.RD_2" Destination="AD.IN2" />
</Request>

DEV1.RES1: <Response ID="98" />

DEV1.RES1: <Request ID="99" Action="CREATE" >
  <Connection Source="AD.OUT" Destination="PUB.SD_1" />
</Request>

DEV1.RES1: <Response ID="99" />

DEV1.RES1: <Request ID="100" Action="CREATE" >
  <Connection Source="FF.Q" Destination="SB.QI" />
</Request>

DEV1.RES1: <Response ID="100" />

DEV1.RES1: <Request ID="101" Action="CREATE" >
  <Connection Source="SB.QO" Destination="AD.QI" />
```

```
</Request>

DEV1.RES1: <Response ID="101" />

DEV1.RES1: <Request ID="102" Action="CREATE" >
  <Connection Source="SB.QO" Destination="PUB.QI" />
</Request>

DEV1.RES1: <Response ID="102" />

DEV1.RES1: <Request ID="103" Action="WRITE" >
  <Connection Source="&#34;225.0.0.2:1026&#34;" Destination="SB.ID" />
</Request>

DEV1.RES1: <Response ID="103" />

DEV1.RES1: <Request ID="104" Action="WRITE" >
  <Connection Source="&#34;225.0.0.1:1025&#34;" Destination="PUB.ID"
/>
</Request>

DEV1.RES1: <Response ID="104" />
```

# Annex C (informative) IEC 61499 Interface Class Specification

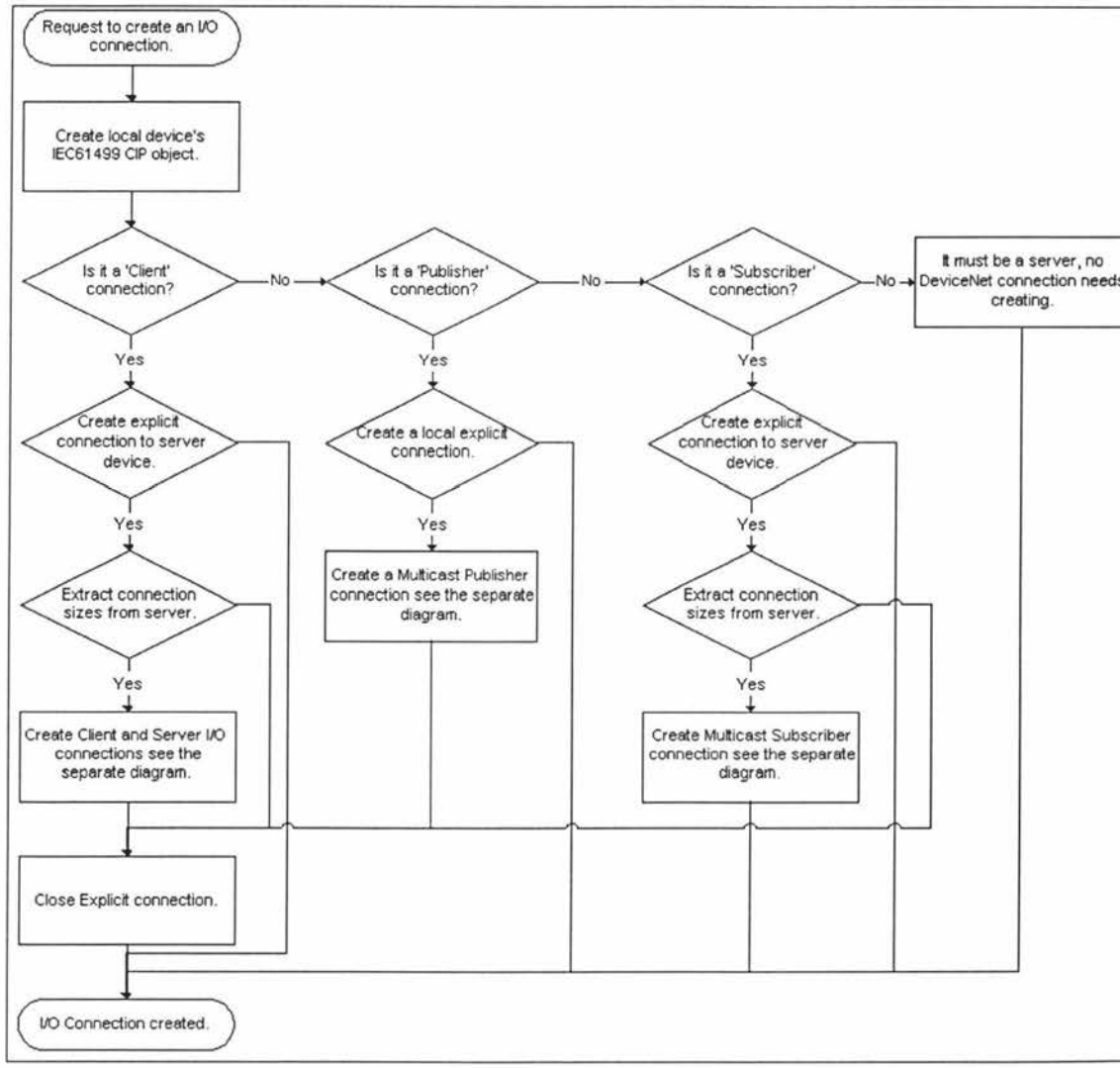
For this thesis this document is in appendix G.

# Annex D (informative) Establishment of Dynamic I/O connections with DeviceNet

This annex is designed to describe DeviceNet peer-to-peer I/O, which is discussed heavily within the DeviceNet specification. The peer-to-peer connections use Dynamic I/O connections, which are discussed in clause 3-3.2 of the ODVA DeviceNet specification. The dynamic I/O communications do not use the "Predefined Master/Slave connection set". To create a dynamic I/O connection an explicit messaging connection is created from the SERVER or SUBSCRIBER function blocks, this will send a 'create service' to the DeviceNet connection class. Using explicit messaging the I/O connection is configured. The flow charts below illustrate the connection procedure.

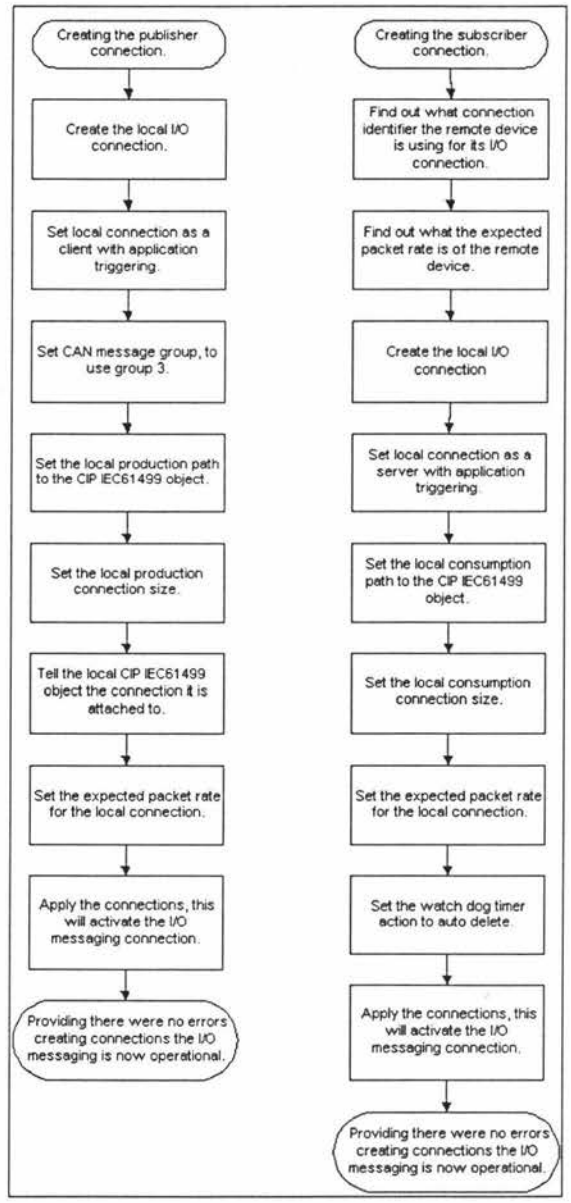
## Creation of an CLIENT/SERVER connection

There are two connections required, each sends data a different direction. The SERVER device will only create an IEC 61499 interface object. The CLIENT device will create an IEC 61499 interface object and the I/O connections. For CLIENT/SERVER connection's CAN message group 1 is used. The transmission trigger shall be application triggering. The diagram below shows connection creation and establishment for CLIENT/SERVER connections.



## Creation of PUBLISHER/SUBSCRIBER connections

The PUBLISHER device will create an IEC 61499 interface object and one dynamic I/O connection, this will be configured to use CAN message group 3 and application triggering. The SUBSCRIBER will create an IEC 61499 interface object that will communicate to the production size parameter and the produced connection object identifier so it can then create an I/O connection with the publisher. The diagram below discusses the creation of the connections.



# Appendix G The IEC 61499 Interface Class

This appendix contains the IEC 61499 interface class specification. This class specification has been written by the author using a similar format to the specification.

## IEC 61499 Interface Object

Class code: 70hex

This object provides an interface for input/output connections to communicate with other IEC 61499 devices. Each communication interface uses another instance of the IEC 61499 interface class.

### Class Attributes

There are no class attributes.

### Instance Attributes

| Attribute ID | Need in Implementation | Access Rule | Name | DeviceNet data Type | Description of Attribute |
|---|---|---|---|---|---|
| 1 | Required | Get | Production data size | UINT | Size of the produced packets. |
| 2 | Required | Get | Consumption data size | UINT | Size of the consumed packets. |
| 3 | Required | Get | Production data | ARRAY | The produced data which is sent over the communication connection. |
| 4 | Required | Get/Set | Consumption data | ARRAY | The consumed data which is sent over the communication connection. |
| 5 | Required | Get/Set | Produced connection id | UINT | The connection instance identifier of the connection which is producing data. |
| 6 | Required | Get/Set | Consumed connection id | UINT | The connection instance identifier of the connection which is consuming data. |

## Semantics:

***Production data size***

This is the size of the produced data connection once the connection has been established. This defaults to zero, resulting in a packet that only contains header information.

***Consumption data size***

This is the size of the consumed data connection once the connection has been established. This defaults to zero, resulting in a packet that only contains header information.

***Production data***

This is a block of data that is sent by the output connection. The data format is shown below.

| Packet length LSB byte | Packet length MSB byte | Transmission identifier byte | Message data |
|---|---|---|---|

***Consumed data***

This is a block of data that is received by the input connection.

| Packet length LSB byte | Packet length MSB byte | Transmission identifier byte | Message data |
|---|---|---|---|

***Produced connection identifier***

The instance identifier of the produced connection, this is a single byte value of the associated connection object identifier.

***Consumed connection identifier***

The instance identifier of the consumed connection, this is a single byte value of the associated connection object identifier.

## Common Services

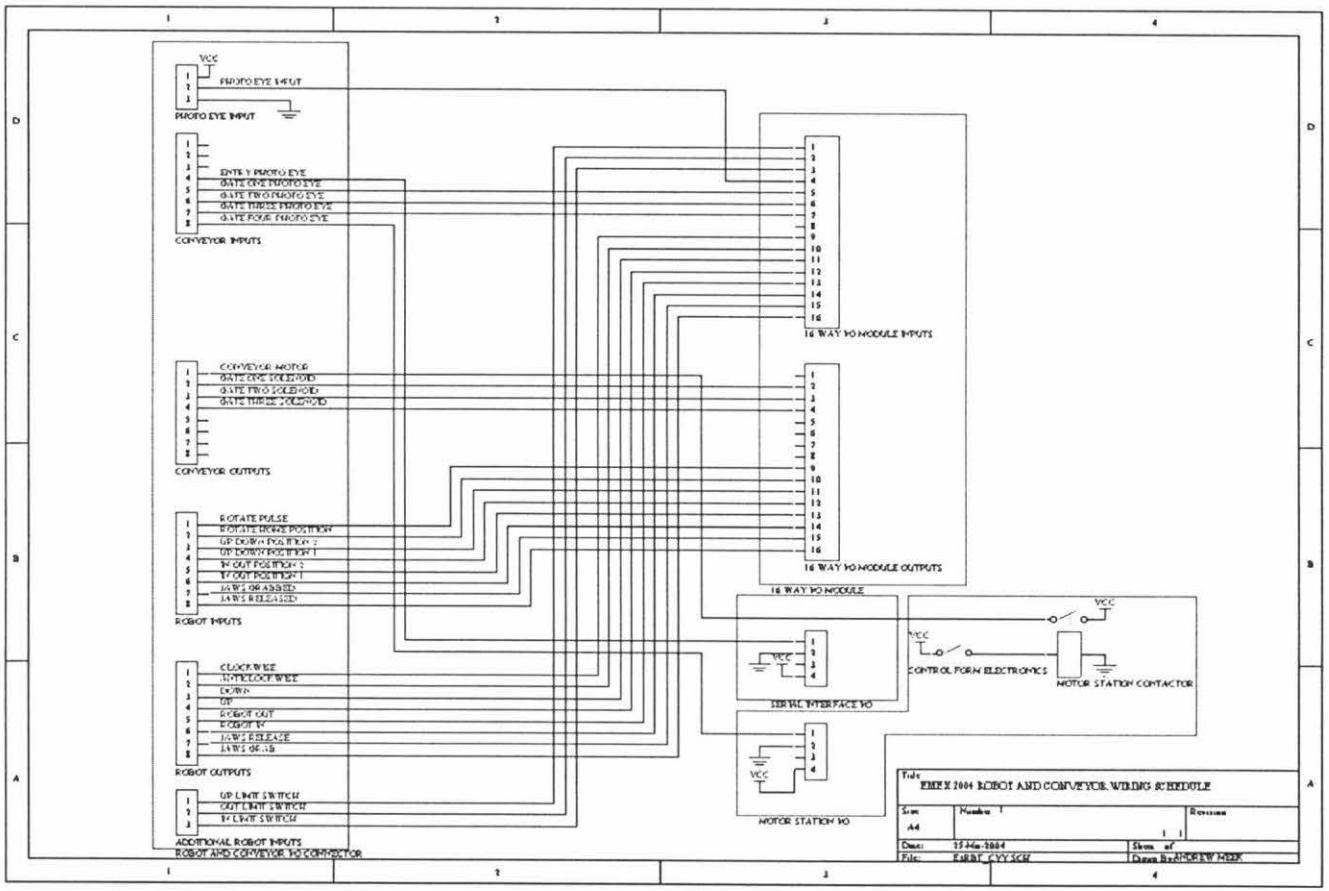
The IEC 61499 interface class provides the following Common Services:

| Service Code | Need in Implementation | Service Name | Description of service |
|---|---|---|---|
| $0E_{hex}$ | Required | Get_Attribute_Single | Returns the value the requested attribute. |
| $10_{hex}$ | Required | Set_Attribute_Single | Set the value of the requested attribute. |

# Appendix H Wiring Schedule for Robot and Conveyor Application

This appendix shows how the robot and conveyor digital I/O interface wiring was connected to the 16 way I/O module, serial communications interface, and motor control station.

**Figure H.1 Wiring schedule for control devices with the robot and conveyor application**



# Appendix I Object Orientation of DeviceNet and Message Formats

This appendix is a reproduction the author's previous work, explaining the format of DeviceNet explicit messaging with regard to DeviceNet object orientation. (Meek, 2003b, p.115-120):

## Object Orientation

"DeviceNet makes use of abstract object modeling to describe:

- The suite of communications services available.
- The externally visible behaviour of a DeviceNet node.
- A common means by which information within DeviceNet products is accessed and exchanged.

A DeviceNet node is modelled as a collection of Objects. An Object provides an abstract representation of a particular component within a product (Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. 1-3).

When mentioning addressing of objects the following terms mean:

- "Object - An abstract representation of a particular component within a product.
- Class - A set of objects that all represent the same kind of system component. A class is a generalisation of an object. All objects in a class are identical in form and behaviour, but may contain different attribute values.
- Instance - A specific and real (physical) occurrence of an object. For example: California is an instance of the object class State. The terms Object, Instance, and Object Instance all refer to a specific Instance.
- Attribute - A description of an externally visible characteristic or feature of an object. Typically attributes provide status information or govern the operation of an Object." (Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. 1-5).

All DeviceNet products implement the following object classes:

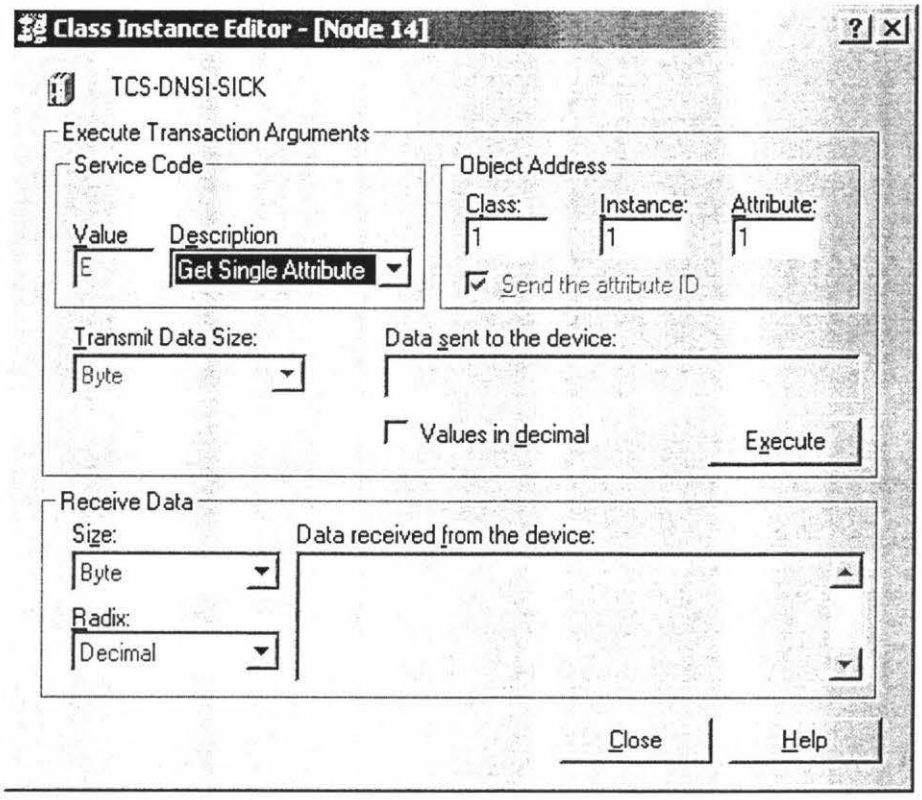
- Identity Class, this stores information about the type of product, product name, revision number and serial number.
- Message Router Class, this is used to route incoming consumption data to the appropriate class that is specified in the message for explicit messaging or the consumed_connection_path for I/O messaging.
- DeviceNet Class, this stores DeviceNet configuration data like MAC ID and baud rate settings. It also sets up the Predefined Master/Slave Connection Set.
- Connection Class: Each connection is an object of the Connection Class. The Connection Class manages the connection looking after the internal processor resources required to keep the connection established.
- There are other classes which are application specific.

When changing or inspecting any device, the classes are addressed as illustrated in the next section.

## DeviceNet Messaging Formats

Addressing of these objects using explicit messaging is often done by the manager of a connection when a connection is being configured, or by a PC running network configuration software. An example of addressing objects by RSNetWorx configuration software is shown below.

**Figure I.1 RSNetWorx Class instance editor tool**




The figure shows the Class Instance Editor tool in RSNetWorx which is a DeviceNet configuration software package. The Class Instance Editor is an interface for a person to write low level DeviceNet messages to a DeviceNet node. The message that would be sent using the example shown in the figure would be: Requesting data from class identifier 1 (Identity Class), instance identifier 1 and attribute identifier 1 (vendor identification code). The service that is to be performed above is Get_attribute_single, the response message will return the vendor identification code. A list of common DeviceNet services is shown below.

**Table I.1 DeviceNet service codes and names**

| Service Code (in hex) | Service Name |
|---|---|
| 00 | Reserved for future DeviceNet use |
| 01 | Get_Attributes_All |
| 02 | Set_Attributes_All Request |
| 03 - 04 | Reserved for future DeviceNet use |
| 05 | Reset |
| 06 | Start |
| 07 | Stop |
| 08 | Create |
| 09 | Delete |
| 0A-0C | Reserved for future DeviceNet use |
| 0D | Apply_Attributes |
| 0E | Get_Attribute_Single |
| 0F | Reserved for future DeviceNet use |
| 10 | Set_Attribute_Single |
| 11 | Find_Next_Object_Instance |
| 12 - 13 | Reserved for future DeviceNet use |
| 14 | Error Response |
| 15 | Restore |
| 16 | Save |
| 17 | No Operation (NOP) |
| 18 | Get Member |
| 19 | Set Member |
| 1A | Insert Member |
| 1B | Remove Member |
| 1C-31 | Reserved for additional DeviceNet Common Services |

(Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. G-2)

Throughout the duration of research the most common services used were:

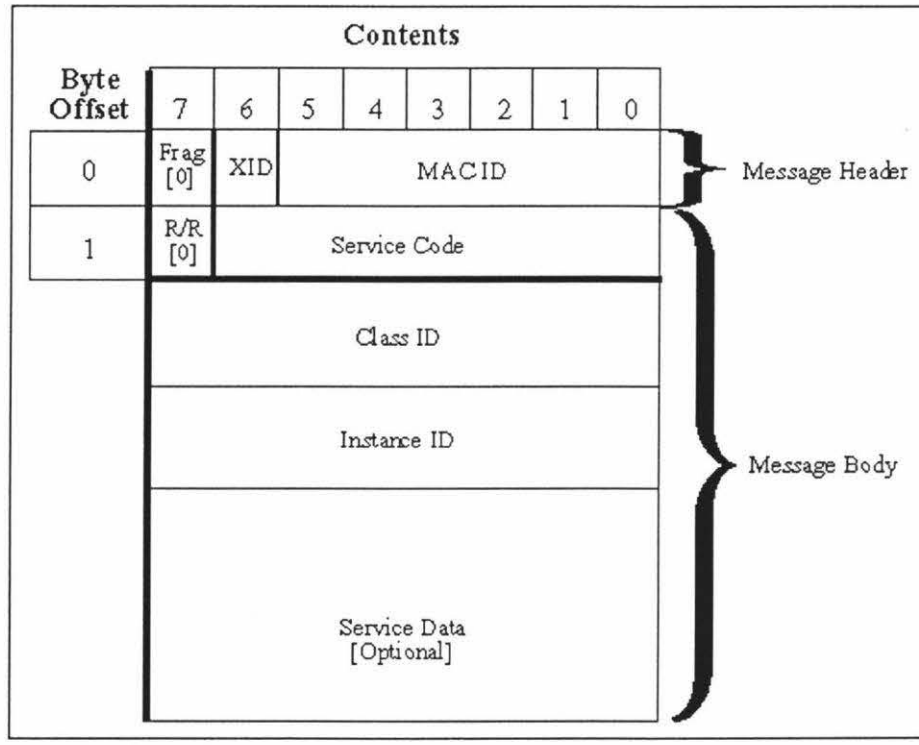
- Get_attribute_single - this requests data from an object.
- Set_attribute_single - this sends data to an object.
- Create - this is used to create an object.
- Delete - this is used to delete an object.
- Reset - this is often used with the Identity Object for re-booting the DeviceNet module or resetting parameters to factory defaults.

For a full definition of the service codes refer to appendix G of the DeviceNet specification. (Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. G-1).

## Standard Non-fragmented Message Formats

All DeviceNet messages are allocated to a class or object. For an I/O message the destination object is specified in the 'connection_path' attributes. For an explicit message, the destination class/object is specified by the first three to five bytes depending on the message format that is configured when the explicit messaging connection was created. The message below illustrates the explicit message format.

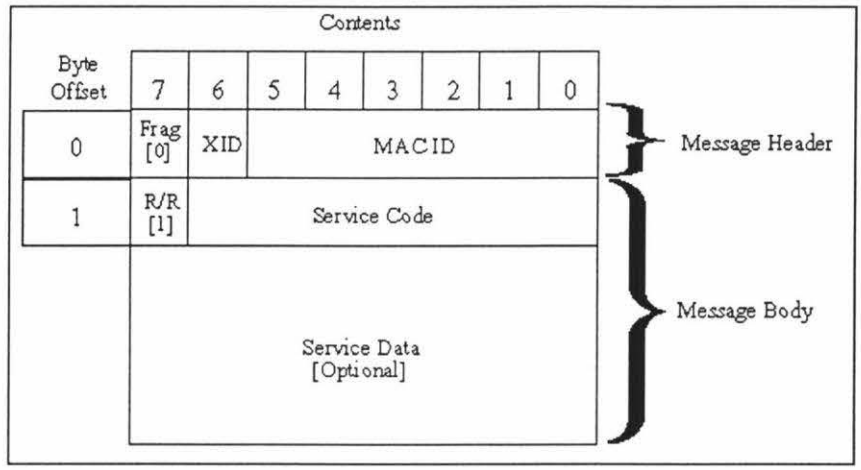
**Figure I.2 Non-fragmented explicit request format**




(Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. 4-19)

If the message is to send an attribute identifier, the first byte of the service data will be the attribute identifier, this is not illustrated in the above figure.

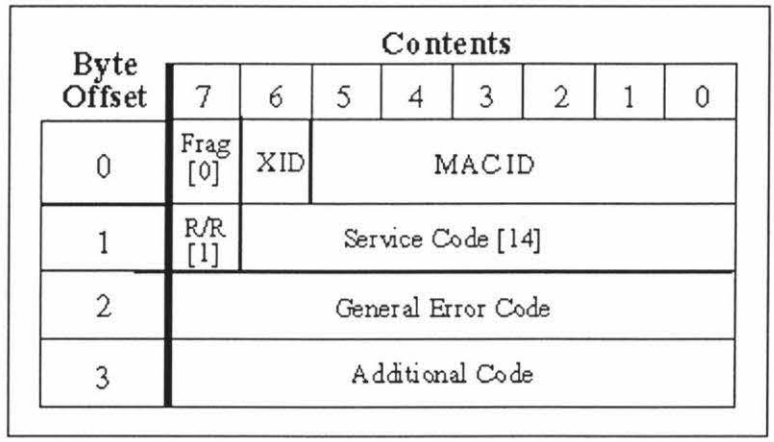
**Figure I.3 Non-fragmented successful response message body format**




(Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. 4-20)

Error response message format is shown below.

**Figure I.4 Error response message**

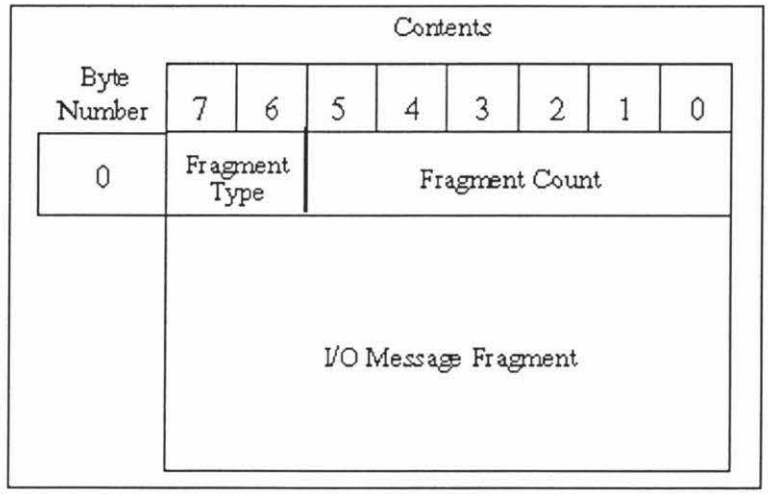



(Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. 4-21)

## Fragmented Message Formats

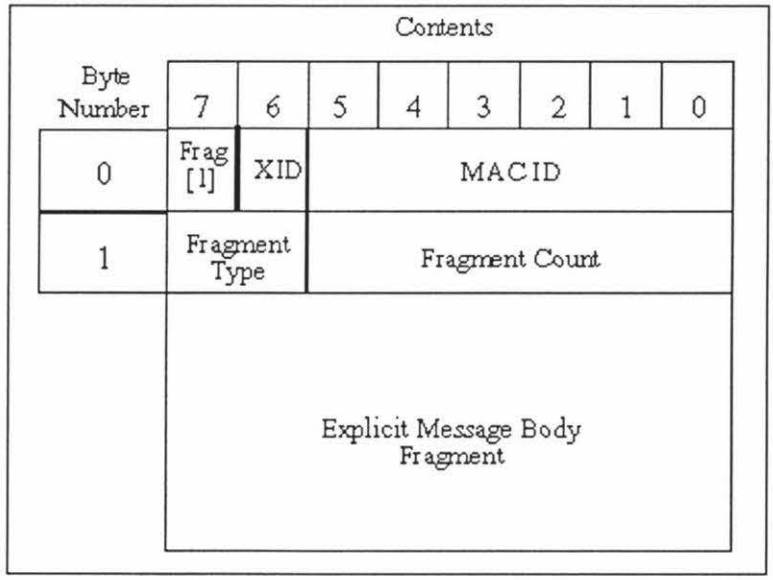
There are two different fragmented message formats, one for I/O and the other for explicit. The contents of the messages are similar to a non-fragmented message except that there is a fragmented header on each message.

**Figure I.5 I/O Message fragment format**




(Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. 4-24)

**Figure I.6 Explicit message fragment format**




(Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. 4-24)

# Appendix J The Connection Class

This appendix explains the properties of the DeviceNet Connection Class and is a reproduction the author's previous work. (Meek, 2003b, p. 110-114):

"The Connection Class allocates and manages the internal resources associated with both I/O and Explicit Messaging Connections. The specific instance generated by the Connection Class is referred to as a Connection Instance or Connection Object. A Connection Object within a particular module actually represents one of the end-points to be configured and 'active' (e.g. transmitting) without the other end-point(s) being present." (Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. 5-5). Each connection object has the following attributes which are accessible via DeviceNet.

**Table J.1 Connection object instance attributes**

| Attribute ID (decimal) | Need In Implementation | Attribute Name | Data Type | Brief Description of Attribute |
|---|---|---|---|---|
| 1 | Required | State | USINT | State of the object |
| 2 | Required | instance_type | USINT | Indicates either I/O or Messaging Connection |
| 3 | Required | transportClass_trigger | BYTE | Defines behavior of the Connection |
| 4 | Required | produced_connection_id | UINT | Placed in CAN Identifier Field when the Connection transmits |
| 5 | Required | consumed_connection_id | UINT | CAN Identifier Field value that denotes message to be received |
| 6 | Required | initial_comm_characteristics | BYTE | Defines the Message Group(s) across which productions and consumptions associated with this Connection occur |
| 7 | Required | produced_connection_size | UINT | Maximum number of bytes transmitted across this Connection |
| 8 | Required | consumed_connection_size | UINT | Maximum number of bytes received across this Connection |
| 9 | Required | expected_packet_rate | UINT | Defines timing associated with this Connection |
| 10 - 11 | N/A | N/A | N/A | Not used. These attribute IDs have been obsoleted and are no longer defined for a Connection Object |
| 12 | Required | watchdog_timeout_action | USINT | Defines how to handle Inactivity/Watchdog timeouts |
| 13 | Required | produced_connection_path_length | UINT | Number of bytes in the produced_connection_path attribute |
| 14 | Required | produced_connection_path | EPATH | Specifies the Application Object(s) whose data is to be produced by this Connection Object. See Appendix I. |
| 15 | Required | consumed_connection_path_length | UINT | Number of bytes in the consumed_connection_path attribute |
| 16 | Required | consumed_connection_path | EPATH | Specifies the Application Object(s) that are to receive the data consumed by this Connection Object. See Appendix I. |
| 17 | Conditional | production_inhibit_time | UINT | Defines minimum time between new data production. This attribute is required for all I/O Client connections, except those with a production trigger of Cyclic. |

(Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. 5-7)

# Definition of the Attributes

## State Attribute

"This attribute defines the current state of the Connection instance." (Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. 5-8). The following states are available:

- Non-existent.
- Configuring.
- Waiting for Connection ID.
- Established.
- Timed out.
- Deferred Delete.

## instance_type Attribute

"This attribute defines the instance type." (Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. 5-8). The following instance types are listed below:

- Explicit messaging.
- I/O messaging.

## transportClass_trigger Attribute

"Defines whether this is a producing only, consuming only, or both producing and consuming connection." (Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. 5-9). The following options are available:

- Direction.
- Client.
- Server.
- Production triggering.
- Cyclic.
- Change of state.
- Application object.
- Transport Class.
- Class 0 - Produce or consume only.
- Class 2.
- Class 3 - Not used by TCS.

Server connections can only transmit data as response messages. This implies that they have to wait for a request message from the client node before they can produce data. Client connections can transmit a request message at an interval depending on what the production trigger and the expected_packet_rate attribute are set to.

## produced_connection_id Attribute

"Contains the Connection ID to be associated with transmissions sent across this connection (if any). This is the value loaded in the CAN identifier field when this connection transmits." (Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. 5-20).

## consumed_connection_id Attribute

"Contains the Connection ID, which identifies messages to be received across this connection (if any). This is the CAN identifier Field value that is associated with the messages this Connection Object receives." (Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. 5-21).

## initial_comm_charateristics Attribute

"Defines the Message Group(s) across which productions and consumptions associated with this Connection occur." (Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. 5-21).

## produced_connection_size Attribute

"For Explicit Messaging Connections, this attribute signifies the maximum number of Message Body bytes that a module is able to transmit across this Connection.

For I/O Connections, this attribute defines the maximum amount of I/O data that may be produced as a single unit across this connection." (Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. 5-23 to 5-24).

## consumed_connection_size Attribute

"For Explicit Messaging Connections, this attribute signifies the maximum number of Message Body bytes that this module is able to receive across this Connection.
For I/O Connections, this attribute defines the maximum amount of data that may be received as a single unit across this connection." (Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. 5-24 to 5-25).

## expected_packet_rate Attribute

"This attribute is used to generate the values loaded into the Transmission Trigger Timer and the Inactivity/Watchdog Timer." (Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. 5-25). The inactivity/watchdog timer times out connections when there is no activity on that connection, its behaviour is defined by the 'watchdog_timeout_action' attribute defined below. The transmission trigger timer is the timer used to trigger cyclic connection transmission.

## watchdog_timeout_action

"This attribute defines the action the Connection Object should perform when the Inactivity/Watchdog Timer expires." (Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. 5-26). The following options are available:

- Transition to timed out.
- Auto delete.
- Auto reset.
- Deferred delete.

## produced_connection_path & produced_connection_path_length Attributes

"The produced_connection_path attribute is made up of a byte stream which defines the Application Object(s) whose data is to be produced by this Connection Object." (Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. 5-27). The produced

_connection_path_length contains the number of bytes that are in the byte stream of the produced_connection_path.

## consumed_connection_path & consumed_connection_path_length Attribute

"The consumed_connection_path attribute is made up of a byte stream which defines the Application Object(s) that are to receive the data consumed by this Connection Object." (Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. 5-27). The consumed_connection_path_length contains the number of bytes that are in the byte stream of the consumed_connection_path.

## production_inhibit_time Attribute

"This attribute is used to configure the minimum delay time between new data production." (Open DeviceNet Vendor Association, Release 2.0 Eratta 5, vol. 1, p. 5-27). This is only used by client connections, if the application requires it. This timer can offer protection for Change of State messages, if a device is toggling Change of State and using all the network bandwidth.

# Appendix K Evaluation of Embedded Java Benchmark Software Tests

The source code of the benchmark software is listed below to show the similarities between the Java and C application. Note that the 'for loops' have been commented out for the second test of this application.

## C Application Source Code

```
#include <sfr62.h>

long targetValue;

//Detects a change of digital input to continue test
void trigChange(void)  {
        static char lastVal = 1;
        while(p7_3 == lastVal);
        lastVal = p7_3;
}

int main( void ) {
        long testValue1 = 0;
        long testValue2 = 0;
        long testValue3 = 0;
        long testValue4 = 0;
        long i;

        // Configure Micro processor i/O
        pd7_2 = 1;
        p7_2 = 1;

        //Find out the I/O access delay
        trigChange();
        p7_2 = 0;       // Turn output pin on
        p7_2 = 1;       // Turn output pin off


        //Loop until input is active
        trigChange();
        //The binary OR test
        p7_2 = 0;
//      for(i=0; i<1000; i++) {
                testValue1 = 0xAAAAAAAA;
                testValue2 = 0x55555555;
                testValue3 = testValue1 | testValue2;
                //Value saved to global to stop complier removing this code during
optimization
                targetValue = testValue3;
//      }
        p7_2 = 1;


        //The binary AND test
        trigChange();
        p7_2 = 0;
//      for(i=0; i<1000; i++) {
                testValue1 = 0xAAAAAAAA;
                testValue2 = 0x55555555;
                testValue3 = testValue1 & testValue2;
                //Value saved to global to stop complier removing this code during
optimization
                targetValue = testValue3;
```

```
//      }
        p7_2 = 1;

        //The binary NOT test
        trigChange();
        p7_2 = 0;
//      for(i=0; i<1000; i++) {
                testValue1 = 0xAAAAAAAA;
                testValue3 = ~testValue1;
                //Value saved to global to stop complier removing this code during
optimization
                targetValue = testValue3;
//      }
        p7_2 = 1;

        //The binary ADD test
        trigChange();
        p7_2 = 0;
//      for(i=0; i<1000; i++) {
                testValue1 = 0xAAAAAAAA;
                testValue2 = 0x55555555;
                testValue3 = testValue1 + testValue2;
                //Value saved to global to stop complier removing this code during
optimization
                targetValue = testValue3;
//      }
        p7_2 = 1;

        //The binary SUB test
        trigChange();
        p7_2 = 0;
//      for(i=0; i<1000; i++) {
                testValue1 = 0xAAAAAAAA;
                testValue2 = 0x55555555;
                testValue3 = testValue1 - testValue2;
                //Value saved to global to stop complier removing this code during
optimization
                targetValue = testValue3;
//      }
        p7_2 = 1;

        //The binary MULTIPLY test
        trigChange();
        p7_2 = 0;
//      for(i=0; i<1000; i++) {
                testValue1 = 0xAAAAAAAA;
                testValue2 = 0x55555555;
                testValue3 = testValue1 * testValue2;
                //Value saved to global to stop complier removing this code during
optimization
                targetValue = testValue3;
//      }
        p7_2 = 1;

        //The binary DIVIDE test
        trigChange();
        p7_2 = 0;
//      for(i=0; i<1000; i++) {
                testValue1 = 0xAAAAAAAA;
                testValue2 = 0x55555555;
                testValue3 = testValue1 / testValue2;
                //Value saved to global to stop complier removing this code during
optimization
                targetValue = testValue3;
//      }
        p7_2 = 1;

        //The binary REMAINDER test
        trigChange();
        p7_2 = 0;
//      for(i=0; i<1000; i++) {
                testValue1 = 0xAAAAAAAA;
                testValue2 = 0x55555555;
                testValue3 = testValue1 % testValue2;
```

```
                //Value saved to global to stop complier removing this code during
optimization
                targetValue = testValue3;
 //     }
       p7_2 = 1;

       //The logical shift left test
       trigChange();
       p7_2 = 0;
//     for(i=0; i<1000; i++) {
               testValue1 = 0xAAAAAAAA;
               testValue2 = 6;
               testValue3 = testValue1 << testValue2;
               //Value saved to global to stop complier removing this code during
optimization
               targetValue = testValue3;
//     }
       p7_2 = 1;

       //The logical shift right test
       trigChange();
       p7_2 = 0;
//     for(i=0; i<1000; i++) {
               testValue1 = 0xAAAAAAAA;
               testValue2 = 6;
               testValue3 = testValue1 >> testValue2;
               //Value saved to global to stop complier removing this code during
optimization
               targetValue = testValue3;
//     }
       p7_2 = 1;

       //The greater than test
       trigChange();
       p7_2 = 0;
//     for(i=0; i<1000; i++) {
               testValue1 = 0xAAAAAAAA;
               testValue2 = 0x55555555;
               if(testValue1 > testValue2)
                       targetValue = testValue1;
//     }
       p7_2 = 1;

       //The less than test
       trigChange();
       p7_2 = 0;
//     for(i=0; i<1000; i++) {
               testValue1 = 0xAAAAAAAA;
               testValue2 = 0x55555555;
               if(testValue2 < testValue1)
                       targetValue = testValue1;
//     }
       p7_2 = 1;

       //The greater than equal test
       trigChange();
       p7_2 = 0;
//     for(i=0; i<1000; i++) {
               testValue1 = 0xAAAAAAAA;
               testValue2 = 0x55555555;
               if(testValue1 >= testValue2)
                       targetValue = testValue1;
//     }
       p7_2 = 1;

       //The less than equal test
       trigChange();
       p7_2 = 0;
//     for(i=0; i<1000; i++) {
               testValue1 = 0xAAAAAAAA;
               testValue2 = 0x55555555;
               if(testValue2 <= testValue1)
                       targetValue = testValue1;
//     }
```

```
        p7_2 = 1;

        //The equal test
        trigChange();
        p7_2 = 0;
//      for(i=0; i<1000; i++) {
                testValue1 = 0xAAAAAAAA;
                testValue2 = 0x55555555;
                if(testValue1 == testValue2)
                        targetValue = testValue1;
//      }
        p7_2 = 1;

        //The not equal test
        trigChange();
        p7_2 = 0;
//      for(i=0; i<1000; i++) {
                testValue1 = 0xAAAAAAAA;
                testValue2 = 0x55555555;
                if(testValue1 != testValue2)
                        targetValue = testValue1;
//      }
        p7_2 = 1;

        //The logical AND test
        trigChange();
        p7_2 = 0;
//      for(i=0; i<1000; i++) {
                testValue1 = 1;
                testValue2 = 1;
                if(testValue1 && testValue2)
                        targetValue = testValue1;
//      }
        p7_2 = 1;

        //The logical OR test
        trigChange();
        p7_2 = 0;
//      for(i=0; i<1000; i++) {
                testValue1 = 1;
                testValue2 = 1;
                if(testValue1 || testValue2)
                        targetValue = testValue1;
//      }
        p7_2 = 1;

        //Standard for loop test
        trigChange();
        p7_2 = 0;
        for(i=0; i<5000; i++) {
                testValue1 = testValue2;
        }
        p7_2 = 1;
}
```

# Java Application Source Code

```
/*
 * Perf_test.java
 *
 * Created on 29 March 2004, 09:46
 */

import cpu4io.*;
import java.io.*;

/**
 *
 * @author  andrewm
 */
public class Perf_test {
    int targetValue;
```

```
    boolean lastInValue = true;

    /**
     * @param args the command line arguments
     */
    public void trigChange(DigitalIn d) {
        try {
            while(lastInValue == d.getIn());
            lastInValue = d.getIn();
        }
        catch (IOException e) {}
    }

    public Perf_test() {
                int testValue1 = 0;
        int testValue2 = 0;
        int testValue3 = 0;
        int testValue4 = 0;
        boolean bValue1 = false;
        boolean bValue2 = false;
        boolean bValue3 = false;
        boolean bValue4 = false;
        int i;
        try {
            DigitalOut DigiOut = new DigitalOut("OUTPUT0_0");
            DigitalIn DigiIn = new DigitalIn("INPUT0_0");
            trigChange(DigiIn);
            //The binary OR test
            DigiOut.setOut(true);
            DigiOut.setOut(false);



            //Loop until input is active
            trigChange(DigiIn);
            //The binary OR test
            DigiOut.setOut(true);
//            i=1000;
//            while(i-- != 0) {
//            for(i=0; i<1000; i++) {
              testValue1 = 0xAAAAAAAA;
              testValue2 = 0x55555555;
              testValue3 = testValue1 | testValue2;
              //Value saved to global to stop complier removing this code during
optimization
              targetValue = testValue3;
//            }
//            }
            DigiOut.setOut(false);


            //The binary AND test
            trigChange(DigiIn);
            DigiOut.setOut(true);
//            for(i=0; i<1000; i++) {
              testValue1 = 0xAAAAAAAA;
              testValue2 = 0x55555555;
              testValue3 = testValue1 & testValue2;
              //Value saved to global to stop complier removing this code during
optimization
              targetValue = testValue3;
//            }
            DigiOut.setOut(false);

            //The binary NOT test
            trigChange(DigiIn);
            DigiOut.setOut(true);
//            for(i=0; i<1000; i++) {
              testValue1 = 0xAAAAAAAA;
              testValue3 = ~testValue1;
              //Value saved to global to stop complier removing this code during
optimization
              targetValue = testValue3;
//            }
```

```
            DigiOut.setOut(false);

            //The binary ADD test
            trigChange(DigiIn);
            DigiOut.setOut(true);
//            for(i=0; i<1000; i++) {
                testValue1 = 0xAAAAAAAA;
                testValue2 = 0x55555555;
                testValue3 = testValue1 + testValue2;
                //Value saved to global to stop complier removing this code during
optimization
                targetValue = testValue3;
//            }
            DigiOut.setOut(false);

            //The binary SUB test
            trigChange(DigiIn);
            DigiOut.setOut(true);
//            for(i=0; i<1000; i++) {
                testValue1 = 0xAAAAAAAA;
                testValue2 = 0x55555555;
                testValue3 = testValue1 - testValue2;
                //Value saved to global to stop complier removing this code during
optimization
                targetValue = testValue3;
//            }
            DigiOut.setOut(false);

            //The binary MULTIPLY test
            trigChange(DigiIn);
            DigiOut.setOut(true);
//            for(i=0; i<1000; i++) {
                testValue1 = 0xAAAAAAAA;
                testValue2 = 0x55555555;
                testValue3 = testValue1 * testValue2;
                //Value saved to global to stop complier removing this code during
optimization
                targetValue = testValue3;
//            }
            DigiOut.setOut(false);

            //The binary DIVIDE test
            trigChange(DigiIn);
            DigiOut.setOut(true);
//            for(i=0; i<1000; i++) {
                testValue1 = 0xAAAAAAAA;
                testValue2 = 0x55555555;
                testValue3 = testValue1 / testValue2;
                //Value saved to global to stop complier removing this code during
optimization
                targetValue = testValue3;
//            }
            DigiOut.setOut(false);

            //The binary REMAINDER test
            trigChange(DigiIn);
            DigiOut.setOut(true);
//            for(i=0; i<1000; i++) {
                testValue1 = 0xAAAAAAAA;
                testValue2 = 0x55555555;
                testValue3 = testValue1 % testValue2;
                //Value saved to global to stop complier removing this code during
optimization
                targetValue = testValue3;
//            }
            DigiOut.setOut(false);

            //The logical shift left test
            trigChange(DigiIn);
            DigiOut.setOut(true);
//            for(i=0; i<1000; i++) {
                testValue1 = 0xAAAAAAAA;
                testValue2 = 6;
                testValue3 = testValue1 << testValue2;
```

```
                //Value saved to global to stop complier removing this code during
optimization
                targetValue = testValue3;
//            }
            DigiOut.setOut(false);

            //The logical shift right test
            trigChange(DigiIn);
            DigiOut.setOut(true);
//            for(i=0; i<1000; i++) {
                testValue1 = 0xAAAAAAAA;
                testValue2 = 6;
                testValue3 = testValue1 >> testValue2;
                //Value saved to global to stop complier removing this code during
optimization
                targetValue = testValue3;
//            }
            DigiOut.setOut(false);

            //The greater than test
            trigChange(DigiIn);
            DigiOut.setOut(true);
//            for(i=0; i<1000; i++) {
                testValue1 = 0xAAAAAAAA;
                testValue2 = 0x55555555;
                if(testValue1 > testValue2)
                    targetValue = testValue1;
//            }
            DigiOut.setOut(false);

            //The less than test
            trigChange(DigiIn);
            DigiOut.setOut(true);
//            for(i=0; i<1000; i++) {
                testValue1 = 0xAAAAAAAA;
                testValue2 = 0x55555555;
                if(testValue2 < testValue1)
                    targetValue = testValue1;
//            }
            DigiOut.setOut(false);

            //The greater than equal test
            trigChange(DigiIn);
            DigiOut.setOut(true);
//            for(i=0; i<1000; i++) {
                testValue1 = 0xAAAAAAAA;
                testValue2 = 0x55555555;
                if(testValue1 >= testValue2)
                    targetValue = testValue1;
//            }
            DigiOut.setOut(false);

            //The less than equal test
            trigChange(DigiIn);
            DigiOut.setOut(true);
//            for(i=0; i<1000; i++) {
                testValue1 = 0xAAAAAAAA;
                testValue2 = 0x55555555;
                if(testValue2 <= testValue1)
                    targetValue = testValue1;
//            }
            DigiOut.setOut(false);

            //The equal test
            trigChange(DigiIn);
            DigiOut.setOut(true);
//            for(i=0; i<1000; i++) {
                testValue1 = 0xAAAAAAAA;
                testValue2 = 0x55555555;
                if(testValue1 == testValue2)
                    targetValue = testValue1;
//            }
            DigiOut.setOut(false);
```

```
            //The not equal test
            trigChange(DigiIn);
            DigiOut.setOut(true);
//            for(i=0; i<1000; i++) {
                testValue1 = 0xAAAAAAAA;
                testValue2 = 0x55555555;
                if(testValue1 != testValue2)
                    targetValue = testValue1;
//            }
            DigiOut.setOut(false);

            //The logical AND test
            trigChange(DigiIn);
            DigiOut.setOut(true);
//            for(i=0; i<1000; i++) {
                bValue1 = true;
                bValue2 = true;
                if(bValue1 && bValue2)
                    targetValue = testValue1;
//            }
            DigiOut.setOut(false);

            //The logical OR test
            trigChange(DigiIn);
            DigiOut.setOut(true);
//            for(i=0; i<1000; i++) {
                bValue1 = true;
                bValue2 = true;
                if(bValue1 || bValue2)
                    targetValue = testValue1;
//            }
            DigiOut.setOut(false);

            //Standard for loop test
            trigChange(DigiIn);
            DigiOut.setOut(true);
            for(i=0; i<5000; i++) {
                testValue1 = testValue2;
            }
            DigiOut.setOut(false);
        }
        catch (IOException e) {}
    }



    public static void main(String[] args) {
        new Perf_test();
    }
}
```

# Glossary

**API** - 'Application Program Interface' a set of library functions/methods for the software developer to use when they are programming software.

**CPU3** - The third generation of embedded DeviceNet motherboard used by the developer company.

**CPU4** - The fourth generation of embedded DeviceNet motherboard used by the developer company.

**EMEX** – 'Engineering Machinery and Electronics Exhibition' an engineering trade show that happens every year in New Zealand.

**FBDK** - 'Function Block Development Kit' the IEC 61499 function block programming software supplied by Rockwell Automation.

**FBRT** - 'Function Block Run-Time' the IEC 61499 function block run-time environment supplied by Rockwell Automation.

**FIFO** - 'First In First Out' buffer. A buffer where the first data that is pushed in is the first data that is popped out of it.

**ICE** - 'In Circuit Emulator' a device that emulates an embedded microprocessor so the software developer can debug the application software on an embedded platform.

**J2ME** - 'Java 2 Micro Edition' the Java method library standard that is used in this research project.

**JDK** - 'Java Development Kit' the Java development suit supplied by Sun Microsystems.

**JDWP** - 'Java Debug Wire Protocol' the communications protocol used between the PC and a remote PC when debugging Java software.

**JNI** - 'Java Native Interface' the C or other language interface for Java programs on the PC platform.

**JPDA** - 'Java Platform Debugger Architecture' this is a Java standard debugger interface to communicate with Java virtual machines for debugging.

**KDWP** - 'KVM Debug Wire Protocol' the communications protocol used between the PC and the target platform when debugging embedded Java software.

**KNI** - 'KVM native interface' the C or other language interface for Java programs on the embedded platform.

**KVM** - The author has heard various definitions of this acronym. Some are 'Kaffe Virtual Machine' or 'Kilobyte Virtual Machine'. It is the Java virtual machine used for this project.

**ODVA** – 'Open DeviceNet Vendors Association' the standard setting body of DeviceNet and Ethernet/IP.

**PLC** - 'Programmable Logic Controller' the central controller that is used with a large number of the existing centralised control architectures with factory automation and control.

**PLD** - 'Programmable Logic Device' a device that can be programmed with logic functions, to simplify hardware design.

**SNAP** - A Java development platform that Dr Jim Christensen of Rockwell Automation uses for IEC 61499 run-time environment development.

**SPI** - 'Serial Peripheral Interface' an interface standard used with embedded microprocessors to communicate with other peripherals. These are typically other chips on the motherboard.

**Virtual Machine** - The byte code interpreter that is used to interpret the Java code.

# References

**Barr, M. (2002, June 14).** Towards a Smaller Java. *Embedded.com.* Retrieved from http://www.embedded.com/shared/printableArticle.jhtml?articleID=9900680

**Bull, J. M., Smith, L. A., Ball, C., Pottage, L., Freeman, R. (2003).** Benchmarking Java against C and Fortran for scientific appliations. *Concurrency and computation: Practice and Experience, No. 15, 417-430.*

**Diedrich, C., Russo, F., Winkel, L., Blevins, T. (n.d.).** Function Block Applications in Control Systems Based on IEC 61804. Retrieved June 25, 2003, from http://www.easydelta.com/keytechnologies/advanced/pdf/iec61804.pdf

**Edgar, A. (2003).** *Evaluation of Embedded Java for IEC61499.* [Internal report for Tait Control Systems Limited]

**EPCC (2005).** *Java Grande at EPCC.* Retrieved April 28, 2005, from http://www.epcc.ed.ac.uk/javagrande/

**Eriksson, H., Penker, M. (1998).** *UML Toolkit.* United States of America: John Wiley & Sons, Inc.

**FBDK. (2003).** *Function Block Development Kit software* [Computer program]. Rockwell Automation Advanced Technology.

**Fox, J. (2002).** FAT System Guide: The Root Directory. Retrieved January 12, 2005, from http://home.freeuk.net/foxy2k/disk/disk6.htm

**Heffernan, D., Murphy, M. (2003).** UDP/IP based distributed control fo a hydraulic valve. *Assembly Automation Volume 23, No. 1, 60-68.*

**Holobloc (2002).** *IEC 61499 Compliance Profile For Feasibility Demonstrations.* [Electronic version]. Retrieved June 24, 2004, from http://www.holobloc.com/doc/ita/index.htm

**Howard, D. M. (1997).** Multithreading in the Java Language. *Embedded.com.* Retrieved June 16, 2003, from http://www.embedded.com/97/fe29710.htm

**IEC TC65/WG6(PT1CD)4. (2003).** *Committee draft - function blocks. Part 1-function blocks.* [Available upon request from Dr Jim Christensen, Rockwell Automation Advance Technology].

**IEC TC65/WG6(PT4PAS)FD. (2002).** *Function blocks. Part 4-Rules for compliance profiles.* [Available upon request from Dr Jim Christensen, Rockwell Automation Advance Technology].

**John, K. H., Tiegelkamp, M. (1995).** *IEC 61131-3: Programming Industrial Automation Systems.* Berlin, Germany: Springer-Verlag.

**Klander, L. (2000).** *Core Visual C++ 6.* Upper Saddle River, New Jersey: Prentice Hall.

**Lewis, R. (2001).** *Modeling control systems using IEC 61499: Applying function blocks to distributed systems.* Cornwall, United Kingdom: T J International.

**Lewis, R. (n. d.).** SearchEng: IEC61131-3 Programming Standard by R. W. Lewis. *SearchEng: Control and instrumentation for engineers by engineers.* Retrieved June 24, 2004, from http://www.searcheng.co.uk/selection/control/Articles/IEC61131/main.htm

**Liron, T. (1999, October).** Enhance your Java application with Java Native Interface (JNI). *Java World.* Retrieved June 24, 2004, from http://www.javaworld.com/jw-10-1999/jw-10-jni_p.html

**Meek, A. R. (2003a).** *Report into Requirements for IEC 61499.* [Internal report for Tait Control Systems Limited].

**Meek, A. R. (2003b).** Distributed Functionality for DeviceNet Communication. *Unpublished postgraduate diploma report.* Wellington, New Zealand: Massey University. [Will not be accessible to public till February 2006].

**Mega-Tokyo. (2003).** *Tell Me About Filesystems.* Retrieved June 9, 2003, from http://www.mega-tokyo.com/os-faq-fs.html

**Meyer, G. (2001).** *Distributed Intelligence Options For Tait Control Systems Second Generation DeviceNet.* [Internal report for Tait Control Systems Limited, compiled by Fernbrook Research Ltd.]

**Open DeviceNet Vendor Association. (Release 2.0 Eratta 5).** *DeviceNet Specification Volume I and II.* Open DeviceNet Vendor Association, Inc.

**Open DeviceNet Vendor Assocation. (2001).** ODVA News, Global Networks Issue 2, Volume 1. (2001). *Reduced Installation Time, Enhanced Troubleshooting: DeviceNet Puts Carter Holt Harvey On a Roll.* Open DeviceNet Vendor Association, Inc.

**Rosenthal, S. (1992, May).** Is C++ the not-ready-for-prime-time embedded language? *SLTF Consulting.* Retrieved June 16, 2003, from http://www.sltf.com/articales/pein/pein9205.htm