University of Southern Queensland
Faculty of Engineering and Surveying

OPC Cycle Time Analyser

A dissertation submitted by

Clement Bollaart

In the fulfilment of requirements of

Courses ENG4111 and 4112 Research Project

towards a degree of

Bachelor of Electronics and Computing Engineering

Submitted: October 2007

Abstract

Variations in equipment cycle times can have a significant effect on the output of modern assembly lines. In an effort to identify and act accordingly when these variations occur, the development of a software application to analyse equipment cycle time via OPC is investigated in this project.

Manufacturing organisations continuously seek improvements in equipment performance and output. A common improvement made is the reduction of equipment cycle time, this provides an increase in output for a relatively small capital outlay. These improvements remain relatively unchecked and variations in cycle time due to numerous factors continue to influence equipment performance.

An equipment cycle time analyser can be used to monitor and report variations in cycle time. The resulting information can be used to predict equipment failures and allow maintenance departments to plan corrective actions. This analyser tool can also be used by engineering departments for analysing equipment cycle time during cycle time improvement projects and new equipment installation.

Currently the development of the cycle time analyser system is at a stage where the cycle time of equipment can be monitored using OPC as a means of communication. Additionally a Microsoft Excel application is available which can import the data recorded by the cycle time analyser software and display a step time diagram of a complete equipment cycle.

University of Southern Queensland

Faculty of Engineering and Surveying

ENG4111 Research Project Part 1 &
ENG4112 Research Project Part 2

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Engineering and Surveying, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Engineering and Surveying or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course "Project and Dissertation" is to contribute to the overall education within the student's chosen degree programme. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Professor Frank Bullen
Dean
Faculty of Engineering and Surveying

Certification

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Clement Christiaan Bollaart

Student Number: 0050029162

Signature

11th October 2007

Date

Acknowledgements

I would like to thank the following people for their assistance and support which have contributed greatly to the success of this project.

Dr. Peng (Paul) Wen (supervisor) for his guidance and constructive criticism given during this project.

My colleagues and managers from Robert Bosch (Australia) for their support, understanding and assistance during the project term.

Finally a special thank you to my wife Susan and daughters Jessica and Megan for supporting my pursuit for knowledge.

Table of Contents

Abstract ii
Limitations of Use iii
Certification iv
Acknowledgements v
Table of Contents vi
List of Figures viii
List of Tables viii
Definitions ix
Chapter 1 Introduction 11
1.1 Project Aim 11
1.2 Project Objectives 12
1.3 Chapter Overview 13
1.4 Chapter Summary 14
Chapter 2 Background 15
2.1 History of Lean Manufacturing 15
2.2 Line Balancing 15
2.3 Trends in Current Manufacturing 17
2.4 Current Cycle Time Data 19
2.5 Real Time Data via OPC 20
2.6 TPM and Predictive Maintenance 21
2.7 Benchmarking 22
2.8 Summary 23
Chapter 3 Methodology 25
3.1 Overview 25
3.2 Design Process Model 25
3.3 Communication with OPC Server 26
3.4 Development of User Interface 26
3.5 Design and Implementation of Reports 27
3.6 Programming Language Selection 27
3.7 Analysis of Cycle Times 28
3.7.1 Machine Capability 28
3.7.2 Simplified Machine Capability 29
3.7.2 Example Tolerance Calculation. 30
3.8 Chapter Summary 31
Chapter 4 System Development and Implementation 32
4.1 Overall Description 32
4.2 OPC Communication 32
4.3 OPC OPCDAAUTO.DLL 33
4.4 User Permission Considerations 34
4.5 DCOM Setting Required 35
4.6 Recording Format 35
4.7 Operation of Cycle Time Analyser Software 36
4.7.1 Connection to OPC Server 37
4.7.2 Create OPC group 38
4.7.3 Add OPC Items to Group 38
4.7.4 Start Recording 39
4.7.5 Cycle Time Display 40

4.7.6 User Instructions 41
4.8 Testing 41
4.9 Visualisation of Recorded Data 42
4.10 Visual Basic Limitations 43
4.11 Problems Encountered 44
4.11.1 OPC TimeStamp 44
4.11.2 DCOM Security Settings 45
4.11.3 SyncRead does not Return Timestamp Data as Array 46
4.11.4 OPC Server Capability 47
4.12 Chapter Summary 47
Chapter 5 Conclusions 48
5.1 Achievement of Objectives 48
5.2 Future Work 49
5.2.1 Statistical Evaluation of Tolerance Times 49
5.2.2 Monitor Multiple Machines 49
5.3 Final Summary 50
References 51
Appendix A 52
Project Specification 52
Appendix B 54
OPC Input Items Example 54
Appendix C 55
Cycle Time Analyser Instruction Manual 55
Appendix D 76
DCOM Setting for windows XP 76
Appendix E 86
OPC Tunneller Quotation 86
Appendix F 88
Visual Basic Code 88
F.1 Code Developed 88

List of Figures

FIGURE 1: HENRY FORD (1919) 15
FIGURE 2: FORD ASSEMBLY LINE 15
FIGURE 3: LINE BALANCING WORKSHEET 16
FIGURE 4: HIGH FLEXIBILITY PRODUCTION LINE 18
FIGURE 5: CYCLE TIME STATISTICS – TYPICAL MANUFACTURING DATA 19
FIGURE 6: PDCA CYCLE 22
FIGURE 7: TYPICAL “ACL” ENTRY 26
FIGURE 8: NORMAL DISTRIBUTION 29
FIGURE 9: MACHINE CAPABILITY PROCESS FLOW 30
FIGURE 10: ARCHITECTURE OF THE CODESYS OPC SERVER 33
FIGURE 11: INTERFACING TO OPC SERVERS 34
FIGURE 12: VISUAL BASIC DATA ARRAY 35
FIGURE 13: TEXT FILE FORMAT 36
FIGURE 14: OPC VISUALISATION 36
FIGURE 15: SERVER CONNECT 37
FIGURE 16: GROUP CONNECT 38
FIGURE 17: ADD ITEMS 39
FIGURE 18: FUNCTION BUTTONS 40
FIGURE 19: RESULTS DISPLAY 40
FIGURE 20: EXCEL REPORT 43
FIGURE 21: CODE FOR MILLISECOND CONVERSION 45

List of Tables

TABLE 1: 50 MACHINE MOVEMENT SAMPLES 30
TABLE 2: OVERALL CYCLE TIME VARIATION 42

Definitions

Client – Computers or applications that employ the services of Servers.

Cm, Cmk – Machine capability index

COM – Component Object Model, which supports communication among objects on different computers.

DCOM – Distributed COM

LSL – Lower Stability Limit

MUDA – Waste (non value added)

OLE – Object Linking and Embedding

OPC – OLE for Process Control

PPM – Parts Per Million

RAD – Rapid Application Development, an incremental software process model that emphasizes a short development cycle.

Server – Device which centrally provides certain services within a network.

TCP/IP – Transmission Control Protocol / Internet Protocol, the set of rules which controls virtually the entire communications over the internet and also applies to most separate networks.

TPM - Total Productive Maintenance

TPS – Toyota Production Systems

UTC – Coordinated Universal Time, UTC-based time is loosely defined as the current date and time of day in Greenwich, England

USL – Upper Stability Limit

LSL – Lower Stability Limit

VLAN – Virtual Local Area Network

S - Standard deviation is a measure of the spread of data in relation to the mean. It is the most common measure of the variability of a set of data. If the standard deviation is based on a sampling, it is referred to as 's'.

Chapter 1 Introduction

1.1 Project Aim

This project aims to design an equipment cycle time analyser which enables equipment to be continuously monitored for deviations from a base cycle time value. The cycle time analyser will be capable of monitoring equipment sub-cycle steps and report which segments have caused the base cycle time to be exceeded.

In order to achieve this, real time data will be collected from equipment using TCPIP based communication and this will be compared against predetermined values for different process steps and functions.

When deviations occur the user is to be notified that the cycle time has been exceeded and they will then have the ability to drill deeper into the cycle time results to determine exactly which step or function of the equipment caused the cycle time exception.

1.2 Project Objectives

To achieve the aim of the project a system design was developed based on developing a number of objectives, the sum of which satisfy the project aim.

The objectives identified are:

- Research possible methods to collect real time data from industrial devices.
- Analyse complete equipment cycle and divide operations into logical steps to be monitored by cycle time analyser.
- Design a software program to record step times from online equipment.
    - Record and store best case example.
- Investigate and define tolerance times allowed for various steps (cylinder movements, barcode reading, screwing operations, vision inspection etc).
- Extend software to check real time data collected from a machine cycle with pre-defined times allowed for steps.
    - User to be notified when current equipment cycle deviates + or - from desired best case example.
    - Step which caused cycle time deviation + or - to be identified.
- Evaluate deviations from best case example, and use data collected to maintain and improve equipment reliability.
- Investigate the potential use of such a monitoring device as a predictive maintenance tool. To continually improve the effectiveness and efficiency of production equipment, as required by ISO/TS16949,

> "Particular requirements for the application of ISO 9001:2000 for automotive production and relevant service part organizations".

These objectives provide the basis for this project and make up the project specification.

In addressing the objectives above, the focus is to develop a software application that is not specific to a particular control system. The software should have the flexibility to be used on any system which provides access to real time variables via an OPC communication interface.

1.3 Chapter Overview

Chapter 2. Background.

This chapter examines the desire of manufacturing organisations to continually make improvements to increase their efficiency. Current manufacturing trends are outlined and the availability of plant floor data collection is established. Finally the need and potential benefits of an OPC cycle time analyser are outlined.

Chapter 3. Methodology.

In this chapter the design methodology is detailed. This includes the selection of a software design model, use of existing infrastructure for communication, user interface requirements, report requirements and analysis potential of data collected.

Chapter 4. System Development and Implementation.

Chapter 4 details the system design, operation and implementation of the developed software. The steps taken to test the software application are detailed and the resolution of problems encountered are also discussed. Additionally how the design

meets the objectives and it's potential to be used as a predictive maintenance tool are outlined.

Chapter 5. Conclusions.

Chapter 6 provides a concluding discussion on the project work completed so far and identifies ideas for future development.

1.4 Chapter Summary

This chapter has outlined the project topic to be discussed in this dissertation. The reasons for undertaking this project have been highlighted and the content of the remaining chapters have been outlined.

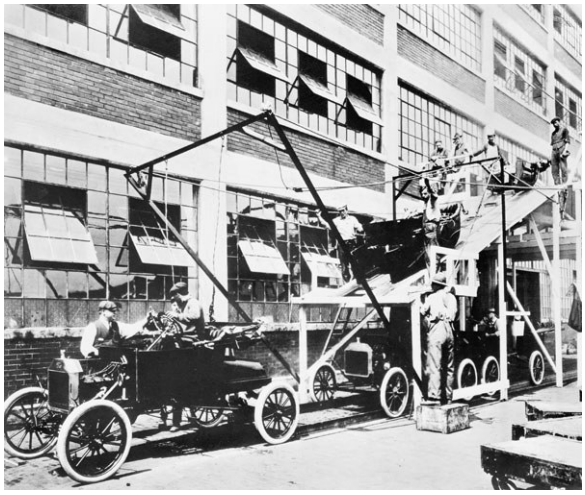
Chapter 2 Background

2.1 History of Lean Manufacturing

The concept of lean manufacturing has been a topic of much discussion since Henry Ford conceived the concept of an assembly line for the production of the Model T Ford back in 1913.

Figure 1: Henry Ford (1919)

The assembly line that was developed by Ford and his team at the Michigan Automobile Factory was a simple one. A motor-drawn rope pulled the chassis pass a number of workstations where workmen performed various operations resulting in a completed automobile at the end of the assembly line.



This simple concept along with other improvements over an 8 year period such as, the introduction of the conveyor belt, reduced the assembly time of a Model T car from 14 hours to 1 hour 33 minutes and enabled a reduction in the selling price from $1000 in 1908 to $360 in 1916.

Figure 2: Ford Assembly line

2.2 Line Balancing

Line balancing is the assignment of operator tasks at workstations to balance operator loops in such a way that the assembly lines output is optimized.

Ever since the introduction of the assembly line by Henry Ford in 1913, line balancing has been an optimization problem of significant industrial importance: the efficiency difference between the optimal and sub-optimal assignment can yield economies (or waste) reaching millions of dollars (Falkenauer 2007).

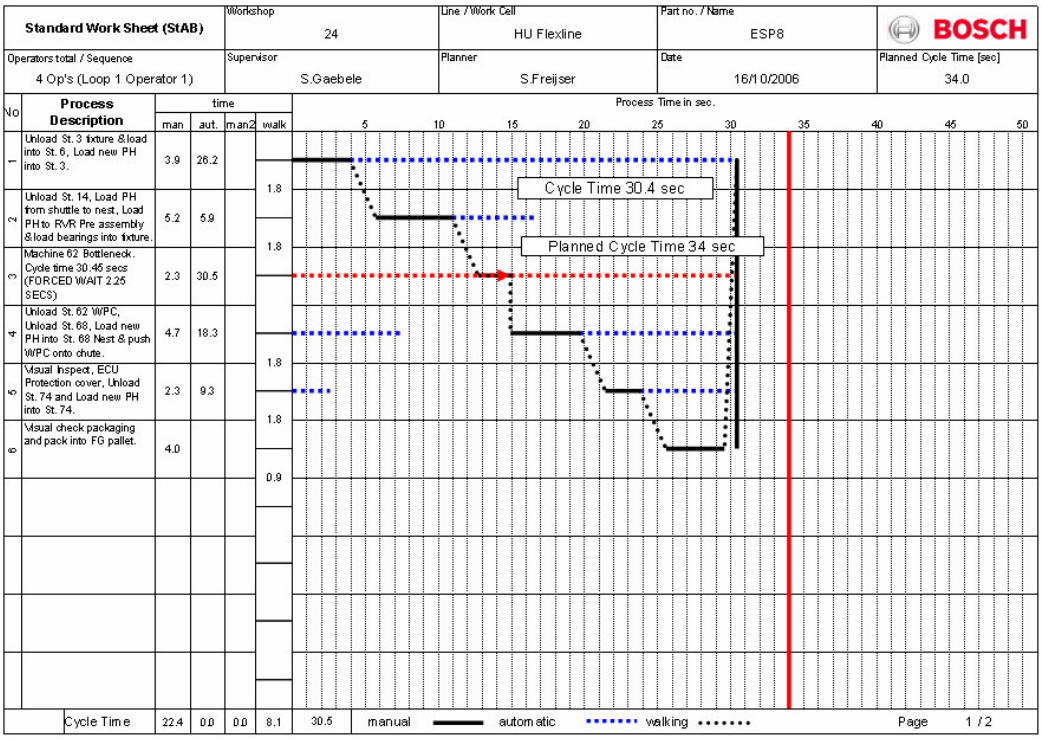


Figure 3: line balancing worksheet

The above worksheet is an example of a typical line balancing exercise. It clearly shows that the quickest operator loop that can be achieved is determined by the bottleneck station. In the example, the actual cycle time of the bottleneck station is 30.5 seconds and is shown as the red dashed horizontal line. If the cycle time of the bottleneck station was to increase for any reason the output of the assembly line would be immediately effected. Cycle time data is one of the most important items for any line balancing project (optimal design).

Assembly line blancing has become a major focus in modern manufacturing processes, the aim of which is to optimize machine and worker patterns to provide maximum output with minimal waste.

2.3 Trends in Current Manufacturing

Manufacturing companies have 2 ways to make a profit.

1. **To increase the selling price.**
2. **To reduce the cost of production.**

As the selling price is generally determined by market conditions most companies turn to reducing the cost of production to maximize profit.

To reduce the cost of production, companies aim to achieve the following goals;

Reduce Cost – By eliminating waste, anything which does not add value to the product. This can relate to motion (time looking and walking) overproduction, inventory (excess stock on hand), waiting time or double handling.

Create a flexible system that can quickly respond to change –

The Vehicle Manufacturing Industry has an ever changing working environment due to, production levels changing with customer demands, absenteeism changes per day and continuous improvements in the manufacturing process.

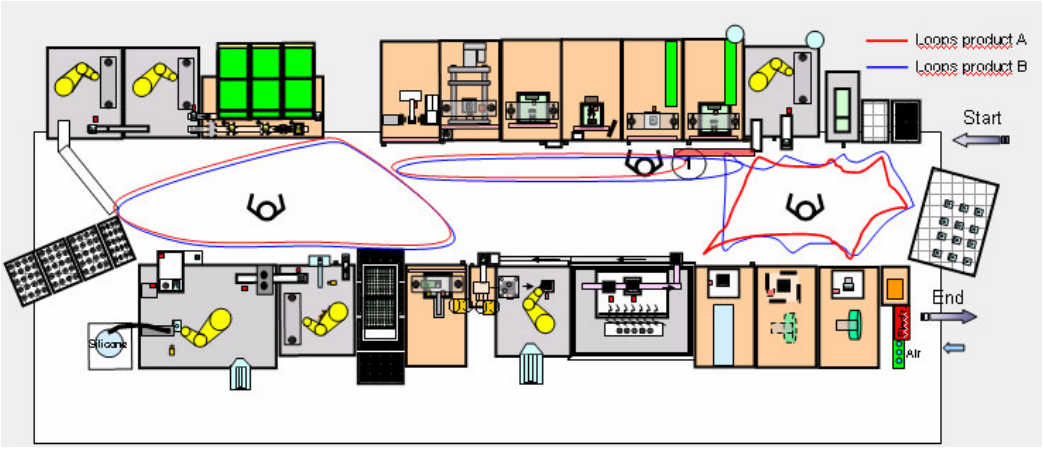


Figure 4: High flexibility production line

Figure 4 above shows a typical high flexibility production line designed to meet the cost and flexibility targets set by most manufacturers. This production line consists of 17 workstations which are shared between 3 operators and produces 2 different product combinations.

Manufacturing companies dedicate large amounts of resources and funds to optimize work place layouts and increase worker efficiency.

This optimization process is based on workstations performing their expected functions within a predetermined cycle time. Any deviation from the expected cycle time will have a large impact on the assembly line output.

The simple example below shows what effect a variation of 1 second in one worker loop can have on the lines output.

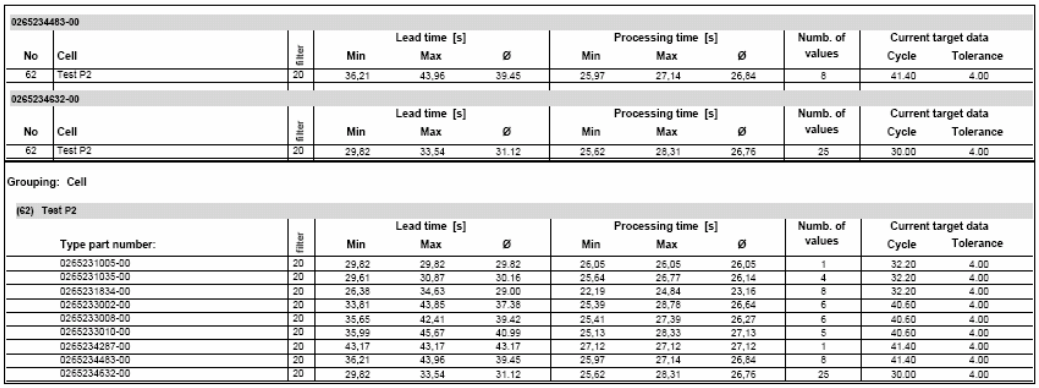
Assuming the line has a cycle time of 30 seconds and all loops have a balanced operator loading. Under normal conditions 120 parts would be produced in one hour. If the bottle neck station in loop 3 has an increase in its cycle time by 1 second the whole line output would drop to 116 parts, resulting in a reduction of output by 3%.

As all 3 loops would be affected by the increase in cycle time, worker efficiency and utilization are also reduced by 3%.

This reduction in output is a perfect example of MUDA (waste) according to TPS (Toyota Production Systems).

2.4 Current Cycle Time Data

The cycle time data provided by most manufacturing information system is usually an average value over a certain period of time per product type as shown in the Figure 5 below.



0265234483-00

No	Cell	filter	Lead time [s] Min	Lead time [s] Max	Lead time [s] Ø	Processing time [s] Min	Processing time [s] Max	Processing time [s] Ø	Numb. of values	Current target data Cycle	Current target data Tolerance
62	Test P2	20	36,21	43,96	39,45	25,97	27,14	26,84	8	41,40	4,00

0265234632-00

No	Cell	filter	Lead time [s] Min	Lead time [s] Max	Lead time [s] Ø	Processing time [s] Min	Processing time [s] Max	Processing time [s] Ø	Numb. of values	Current target data Cycle	Current target data Tolerance
62	Test P2	20	29,82	33,54	31,12	25,62	28,31	26,76	25	30,00	4,00

Grouping: Cell

(62) Test P2

Type part number:	filter	Lead time [s] Min	Lead time [s] Max	Lead time [s] Ø	Processing time [s] Min	Processing time [s] Max	Processing time [s] Ø	Numb. of values	Current target data Cycle	Current target data Tolerance
0265231005-00	20	29,82	29,82	29,82	26,05	26,05	26,05	1	32,20	4,00
0265231035-00	20	29,61	30,87	30,16	25,64	26,77	26,14	4	32,20	4,00
0265231834-00	20	26,38	34,63	29,00	22,19	24,84	23,16	8	32,20	4,00
0265233002-00	20	33,81	43,86	37,38	25,39	28,78	26,64	6	40,60	4,00
0265233008-00	20	35,65	42,41	39,42	25,41	27,39	26,27	6	40,60	4,00
0265233010-00	20	35,99	45,67	40,99	25,13	28,33	27,13	5	40,60	4,00
0265234287-00	20	43,17	43,17	43,17	27,12	27,12	27,12	1	41,40	4,00
0265234483-00	20	36,21	43,96	39,45	25,97	27,14	26,84	8	41,40	4,00
0265234632-00	20	29,82	33,54	31,12	25,62	28,31	26,76	25	30,00	4,00

Figure 5: Cycle time statistics – typical manufacturing data

The data provided in such reports is historical in nature and not sufficient for a real time analysis of an equipments performance. Additionally the data provided is insufficient to enable analysis of sub cycles of the equipment. The processing time in the above statistics varies from a minimum of 22.19 seconds to a maximum of 26.77 seconds for the part number 0265231xxx product family, a variation of 4.58 seconds. Analysis of equipment sub-cycles is not possible from the currently available data and the variation of 4.5 seconds remains unexplained.

2.5 Real Time Data via OPC

In recent years automation technology has developed to a level that allows seamless exchange of information across plant and enterprise networks, (Shimanuki 1999) allowing access to real time data from Automation devices.

OPC stands for **O**LE for **P**rocess **C**ontrol (Matrikon 2005). OPC is one of the technologies that allows the exchange of data between automation devices and it significantly reduces the time, cost and effort required to write custom interfaces to different intelligent devices in use today. OPC provides a standard scalable way to collect and archive critical IT asset data and deliver it to decision makers so that it can be acted on in a timely manner (Murphy 2006).

OPC Data Access or OPC – DA is the OPC specification that is used to read and write real time data exclusively (Kondor 2007). It is a published specification that is being adopted by an increasing number of manufacturers in the automation industry. It sets out how data should be structured and allows devices from different vendors to communicate. OPC is based on Microsoft's COM and DCOM technologies. The OPC foundation, a non-profit international organization made up of hardware and software companies, is responsible for establishing and maintaining the specifications. The specification states that each data point shall include three attributes: a value, a quality and a time stamp. The time stamp reflects the time at which the server knew the corresponding value was accurate (OPC Interface Standard).

OPC performance is more than adequate for most dedicated and distributed applications running on commonly available hardware (Liu, J 2005). Studies have reported OPC servers supplying 20,000 values per second to 4 clients with only 10 percent CPU load. (Chisholm, A, 1998)

2.6 TPM and Predictive Maintenance

Total Productive Maintenance (TPM) is a maintenance philosophy that is associated with the lean manufacturing model (Carreira 2004). Lean production models are based on reducing work in progress (WIP) which requires a balance of operations, continuous material flow and minimal variation; some would call this predictability (Carreira 2004). Predictability would imply that equipment cycle time is stable and consistent over a extended period of time.

As maintenance departments move towards a predictive maintenance strategy, accurate and timely information on the state of assets can be used to detect and correct impending problems before they become catastrophic failures.

The ISO/TS 16949 defines predictive maintenance as: activities based on process data aimed at the avoidance of maintenance problems by the prediction of likely failure modes.

> Section 7.5.1.4 “Preventive and predictive maintenance stipulates that the organization **shall** utilize predictive maintenance methods to continually improve the effectiveness and efficiency of production equipment”.

The aim of any data collection exercise is to get the right data, to the right people, at the right time. Invariably this should have the desired impact on the bottom line.

2.7 Benchmarking

Benchmarking is the process of determining who is the very best, who sets the standard, and what that standard is (Reh 2007).

Benchmarking is another tool which is used by organizations to improve their quality and or output. It follows a similar approach to the PDCA (Plan Do Check Act) continuous improvement cycle.

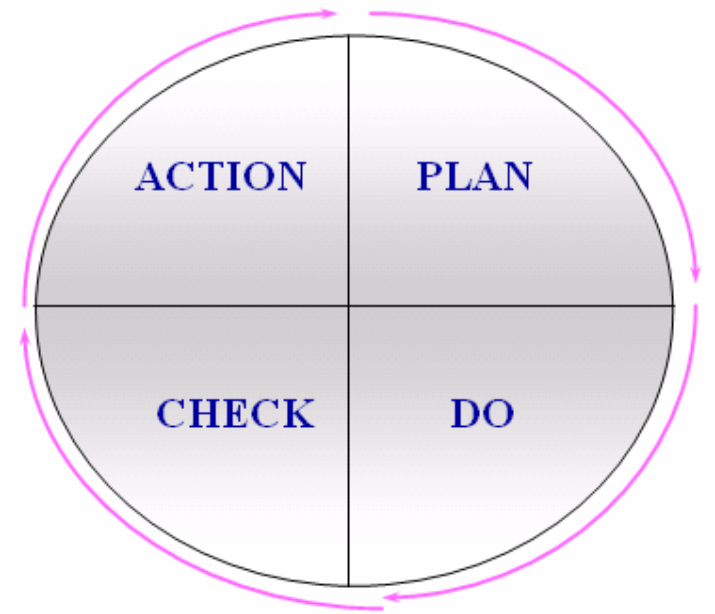


Figure 6: PDCA Cycle

The steps involved in a benchmarking project are:

1. Determine who is the Best. (Plan)
2. Determine how good they are. (Do)
3. How do we get that good. (Check)
4. Implement improvements based on results. (Act)

Any cycle time reduction project can also be considered a benchmarking activity with the following steps.

1. Determine the optimum cycle time. (Plan)
2. Design processes and workflow to optimize cycle time. (Do)
3. Continually monitor cycle time. (Check)
4. Implement improvements based cycle time results. (Act)

Step 3 of the above project requires some form of continual monitoring so that the cycle time achieved by the project is maintained. This is an Ideal application for some form of cycle time analyser.

Equipment processing time (cycle time) or sometimes referred to as *throughput* is a major factor that can effect the output of an assembly line. Equipment cycle time remains largely unchecked in a lot of manufacturing organizations and can have variations of up to a couple of seconds before the effects are noticed as a reduction in output. Each year companies spend large amounts of money to improve and increase the productivity of their equipment but spend little to monitor and "benchmark" their gains.

2.8 Summary

History has shown that manufacturing organizations are continually working to improve assembly line quality and output. Line balancing, TPM, Predictive maintenance and benchmarking are some of the methods used to achieve improvements within these organizations.

Once improvements have been made some form of monitoring should be implemented to ensure that the gains achieved are maintained well into the future. An equipment cycle time analyser would provide a means to monitor improvements that have had a positive effect on equipment cycle time.

An equipment cycle time analyser should be capable of recording the overall and sub - cycle processing times from automated assembly equipment and report any deviations to the user. This data can then be used to investigate the cause of the error and assist with a fast resolution of the problem.

Chapter 3 Methodology

3.1 Overview

A number of key elements need to be addressed in the design of the cycle time analyser software. These elements are:

1. Design process model
2. Communication with OPC server
3. Development of user interface
4. Design and implementation of reports
5. Programming language selection
6. Analysis of cycle times

The Methodology adopted for these elements are discussed in this chapter.

3.2 Design Process Model

Due to the tight timelines and need for concurrent development, required by this project, a RAD (Rapid Application Development) model was chosen for the project. The RAD model allows for concurrent modelling, development and construction phases to occur during the life cycle of the project.

The Development activities will be split into the following concurrent tasks:

1. Communication with OPC server
2. Development of User interface
3. Design and implementation of reports
4. Analysis of equipment cycle times

3.3 Communication with OPC Server

This task involves the investigation and set up of client and server devices to enable the exchange of OPC data using existing Ethernet infrastructure.

The existing network structure has been designed to operate sensible control systems and test equipment for real time measurements without any interference or disturbance coming from local area networks and other data traffic. The system is implemented with a separate VLAN / subnet utilising a dedicated 3 layer switch. This provides a better split in subnets and a more secure separation from the core-layer 3 switches of the network. Figure 7 is an example of an access control list entry.

10.23.79.0-127	partial subnet	<--->	10.23.27.29	Peacy C.Bollaart

Figure 7: Typical "ACL" Entry

The production network is also protected with access control lists to limit incoming and outgoing traffic only to authorised users.

3.4 Development of User Interface

The user interface to be provided should be easy to use. The choice of programming language to be used for the project was also based on the need for a graphical user interface. The user interface should provide an intuitive means to connect and monitoring points to the application. Results should be displayed as a numerical value in addition to a go, no-go indication.

3.5 Design and Implementation of Reports

To be able to analyse the results collected during the monitoring process some method is required to visualise and compare the results obtained. As Microsoft Excel has extensive built in functions to display graphs and manipulate the associated data, it was decided that the implementation of the cycle time analyser should incorporate an excel component for reporting. The reporting function must have the ability to display all 16 recorded signals on one graph so the relationship between the signals can be identified. Additionally there should be the possibility to overlay a reference curve over the last recorded curve so that variations in signal times can be compared. Excel's standard functionality will provide useful functions such as the ability to zoom into a section of the chart by changing the axis scale.

3.6 Programming Language Selection

Visual Basic was selected for the software implementation of this project. This language was chosen due to the large amount of supporting documentation and programming examples identified during the literature review. Numerous examples and a wide selection of reference material is available to assist in the development of a software program with respect to OPC connectivity. Additionally the OPC foundation provides an Automation wrapper DLL suitable for use with Visual Basic. Visual Basic also provides an efficient environment for developing software applications requiring a graphical user interface (GUI).

3.7 Analysis of Cycle Times

3.7.1 Machine Capability

A machine capability study refers to a short term study with the sole aim of discovering the machine specific effects on the production process. These principles can be extended to determine the allowable limits that should be applied to the recorded values of cycle time to ensure stable monitoring results.

The aim of a machine capability study is to reach a conclusion about the behaviour of machine (in control or not?). The following formula is used for the calculation of capability.

$$C_m = \frac{T}{6 \bullet s_{total\backslash}}$$

Where C_m = Capability index

T = Tolerance = USL-LSL

s_{tolal} = total standard deviation

The capability formula above can be transformed to derive the allowable tolerance times based on a required capability index,

$$T = C_m \bullet 6 \bullet s_{total}$$

The Tolerance value needs to be divided by 2 as the cycle time analyser uses a +/- tolerance value.

Figure 8 shows the relationship between a calculated Cmk value and a fraction nonconforming, a Cmk = 1.33 corresponds to 32ppm non-conforming and a Cmk = 1.67 corresponds to 0.03ppm non-conforming.

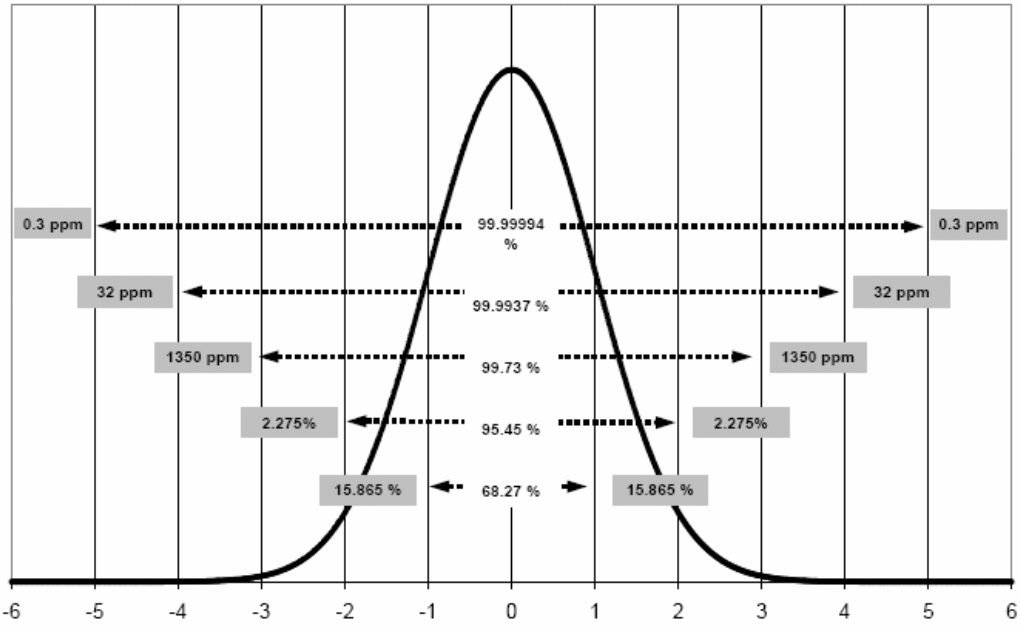


Figure 8: Normal Distribution

3.7.2 Simplified Machine Capability

As capability studies can be very time consuming when considering the large number of values to record, a simplified approach can be adopted to derive tolerance times. Using this approach a reduced number of samples can be used to calculate tolerance times provided a capability index of 2 is applied.

If this approach is used the resulting tolerance time should be subject to a reality check to confirm it's suitability to detecting variations in cycle time.

The simplified approach should only be used on sub cycles with minimal variation otherwise the assumed capability of 2 will result in unrealistically large tolerance times.

The simplified approach is based on the flow chart shown in Figure 9.

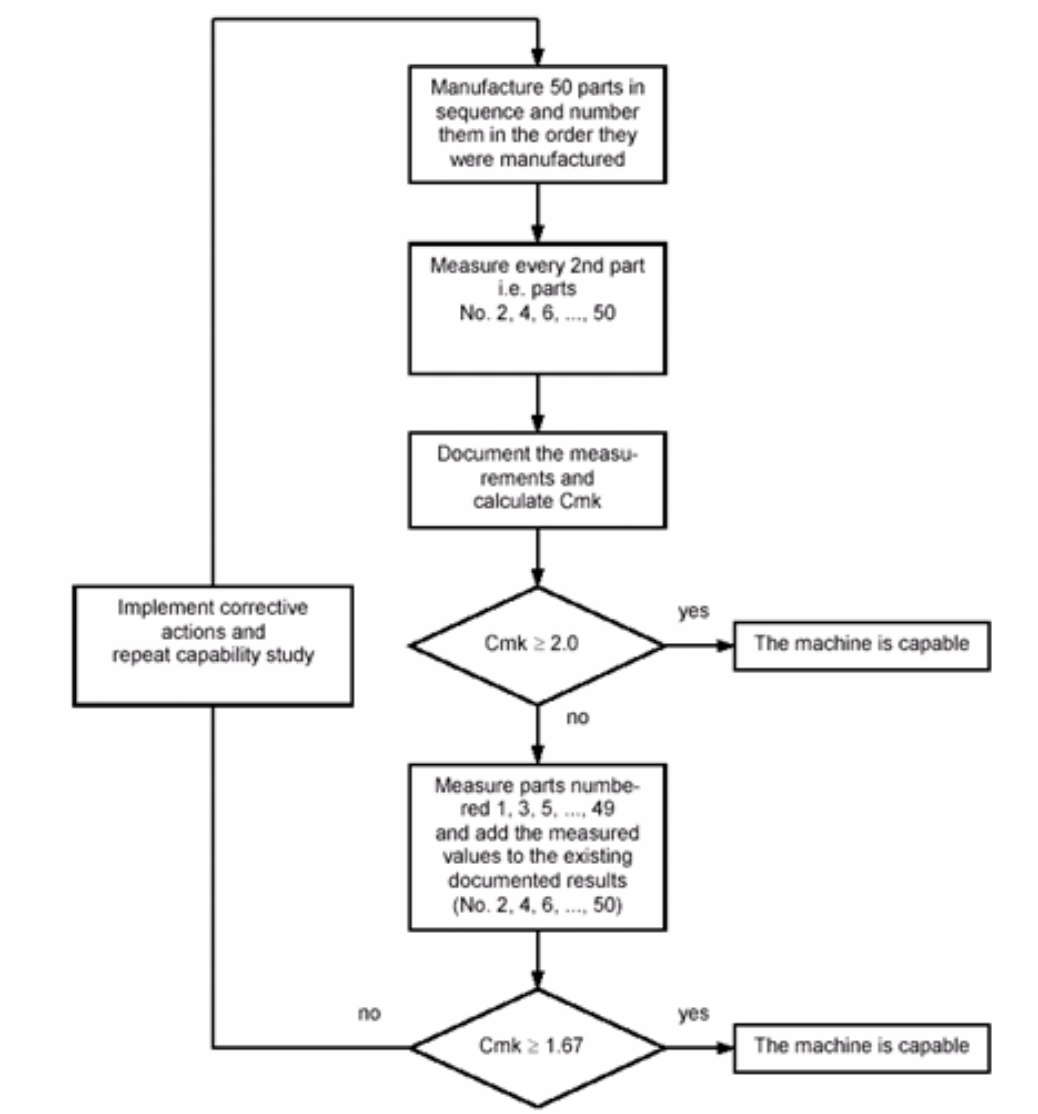


Figure 9: Machine capability process flow

3.7.2 Example Tolerance Calculation.

Table 1 contains 50 Samples taken for a machine press movement.

1	2	3	4	5	6	7	8	9	10
2.353	2.353	2.353	2.353	2.353	2.353	2.353	2.353	2.353	2.353
2.400	2.400	2.400	2.400	2.400	2.400	2.400	2.400	2.400	2.400
2.268	2.268	2.268	2.268	2.268	2.268	2.268	2.268	2.268	2.268
2.400	2.400	2.400	2.400	2.400	2.400	2.400	2.400	2.400	2.400
2.268	2.400	2.268	2.400	2.268	2.400	2.268	2.400	2.268	2.400

Table 1: 50 Machine movement samples

Calculated $s_{total} = 0.058$

Now, $T = C_m \cdot 6 \cdot S_{total} = 1.67 \cdot 6 \cdot 0.058 = 0.58$

This value describes the total tolerance and needs to be divided by 2 to specify an upper and lower limit values.

Tolerance = 0.58 / 2 = 0.29 seconds

In this example 0.29 seconds should be entered into the tolerance field and the average of the 50 measurements, 2.351 seconds should be entered as the base value.

3.8 Chapter Summary

The main issues in the design of the cycle time analyser system will be addressed by the selection of programming language and appropriate communication to provided a cost effective and functional application. Issues that arise due to unforseen circumstances will need to be reviewed as they occur and appropriate actions put in place to resolve these issues.

Chapter 4 System Development and Implementation

4.1 Overall Description

The development and implementation of the cycle time analyser application involved the following main activities:

Connection of the equipments OPC server to Visual Basic client application.

DCOM settings to allow data exchange between server and client.

Programming of client software application.

These activities and associated tasks are discussed in this chapter.

4.2 OPC Communication

OPC is a standardised interface that allows access to process data. It is based on Microsoft's COM/DCOM which has been extended to meet the needs of data access in the automation process where it is mostly used to read and write data from computer based control systems. Typical OPC clients are visualisations and programs for recording operational data.

Due to the features of DCOM it is possible to gain access to an OPC server that is installed on a remote PC. Another advantage of the adoption of COM technology by OPC is programming language independence.

OPC was chosen as the communication medium for this project due to it's ability to support multiple clients and it's focus on providing process data to client programs.

Figure 10 shows the system architecture for typical OPC systems highlighting the ability of the OPC server to communicate with multiple clients.

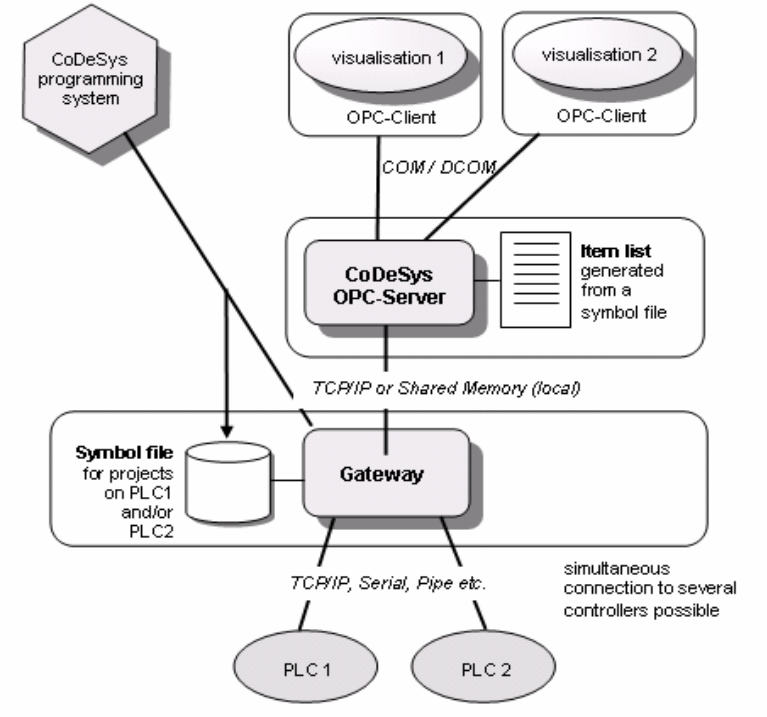


Figure 10: Architecture of the CoDeSys OPC server

4.3 OPC OPCDAAUTO.DLL

A common way is needed for automation applications to access data from field devices or databases. The OPC Data Access Automation defines a standard way by which automation applications can access process data.

Figure 11 shows an OPC client utilising the “wrapper” DLL to call into an OPC Server. The wrapper translates between the custom interface provided by the server and the automation interface desired by the client.

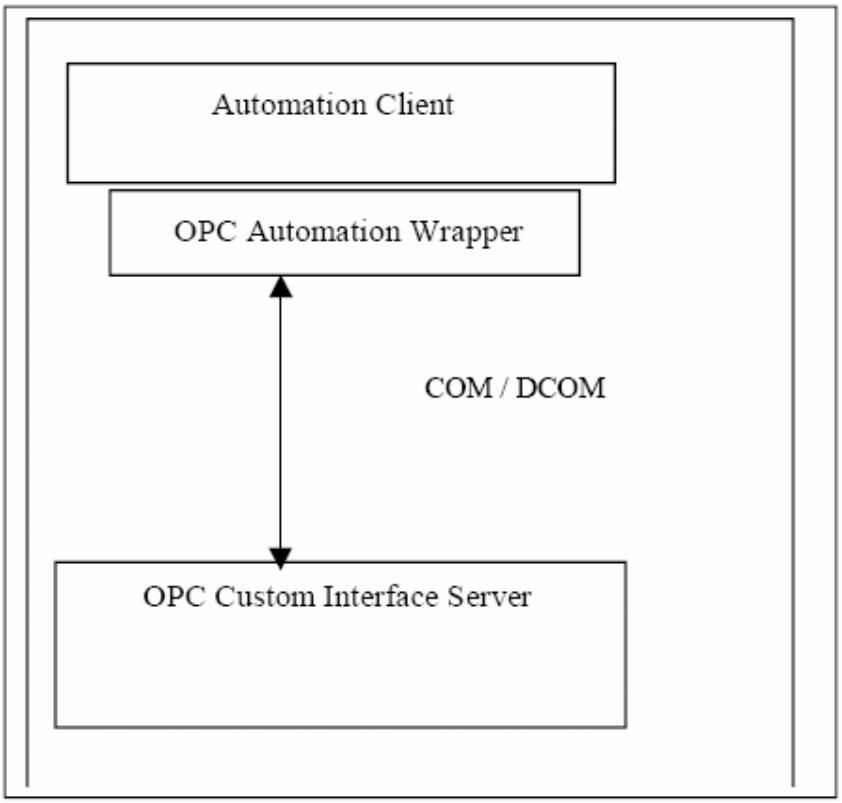


Figure 11: Interfacing to OPC servers

The OPC foundation provides a sample of the Data Access Automation interface for the foundation members use in providing an Automation interface to OPC data access custom interfaces. The sample provided has been used in the implementation of this client application.

4.4 User Permission Considerations

To allow communication between two computers each machine needs to be set up so that they have permission to access each other. This is a two way street. The client must have permissions to access the machine with the OPC server and visa versa. If you do not have permissions set to allow communication in both ways, then all attempts to establish communication will be unsuccessful. For the purposes of this project user permissions for DCOM have been opened up to allow access to all users. For security purposes it is recommended that once communication has been

established and testing completed that user permissions should be reviewed to only allow the necessary users access.

4.5 DCOM Setting Required

See Appendix C for necessary DCOM settings.

4.6 Recording Format

The cycle time application allows for 16 discrete signals to be monitored.

Each signal is sampled at a 50ms interval with a set number of 670 samples per signal. This resolution will allow for an overall equipment cycle time of approximately 33.5 seconds to be recorded. The individual samples are temporarily stored an array as shown in Figure 12 before being transferred to a text file in a tab delimited format.

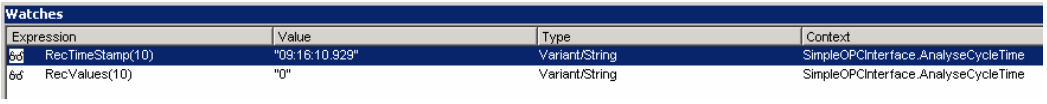


Figure 12: Visual Basic data array

The text file is used to upload the recorded data into Microsoft Excel for the generation of displacement/time diagrams.

Figure 13 shows the format used for the tab delimited text file.

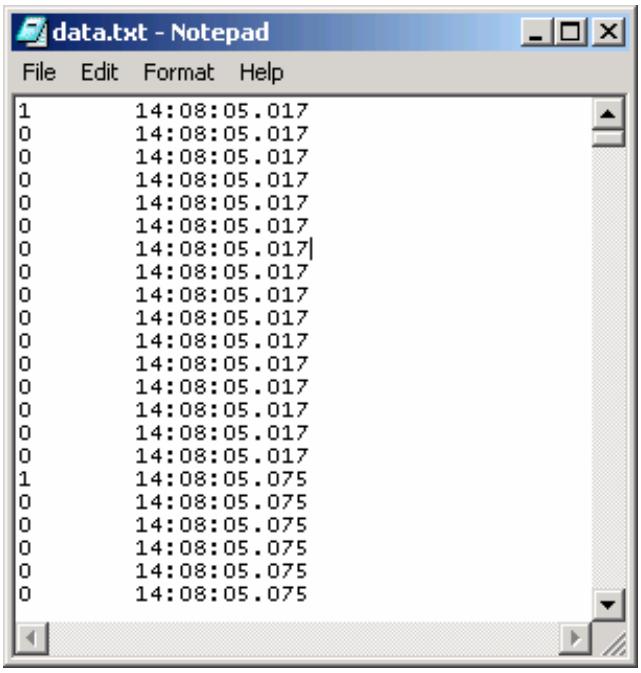


Figure 13: Text file format

4.7 Operation of Cycle Time Analyser Software

Figure 14 shows the main screen developed for the cycle time analyser application.

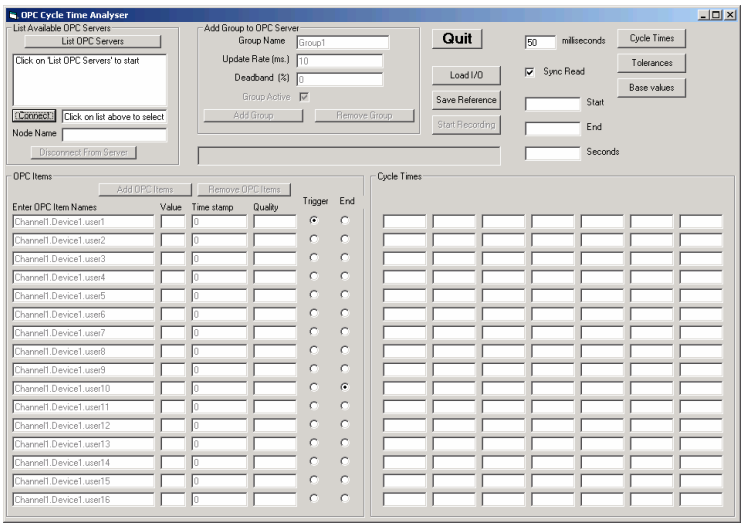


Figure 14: OPC visualisation

It consists of a number of sections:

1. Connection to OPC server
2. Create OPC Group
3. Add OPC items to group
4. Start recording
5. Cycle time display

These sections are described in more detail below.

4.7.1 Connection to OPC Server

The first step in the operation of the cycle time analyser software is to establish connection to the selected automation controller and select the required OPC server. This is achieved by entering the required node name (IP address or computer name) and then pressing the “List OPC Servers” button. A list of available OPC servers will then be displayed. Select the required OPC server and click the “Connect button” Once Connection to the OPC server has been established the “Add group” button will become active. Figure 15 shows the server connect section of the cycle time analyser application.

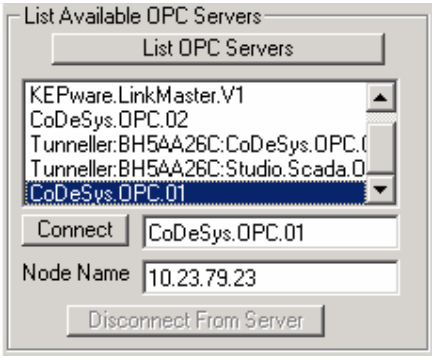


Figure 15: Server connect

4.7.2 Create OPC group

Type the name of the OPC group to be added and click the “Add Group” button. Figure 16 shows the Add Group to OPC Server section.

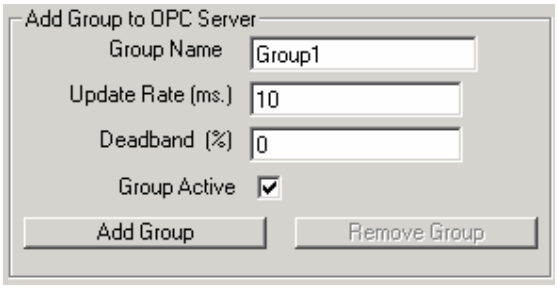


Figure 16: Group connect

4.7.3 Add OPC Items to Group

OPC items can now be added, these represent the input, output or internal memory locations that should be monitored. These can be typed directly into the 16 fields provided or uploaded from a text file using the “Load I/O” button.

Appendix B shows an extract from the target equipments OPC variables list. OPC items need to be entered in the item fields using the same naming convention as displayed in this extract.

Trigger points can also be selected:

Trigger - determines the signal which will start the recording process, positive edge trigger.

End - determines which signal is to be used to calculate the overall cycle time.

Figure 17 shows the OPC Items section.

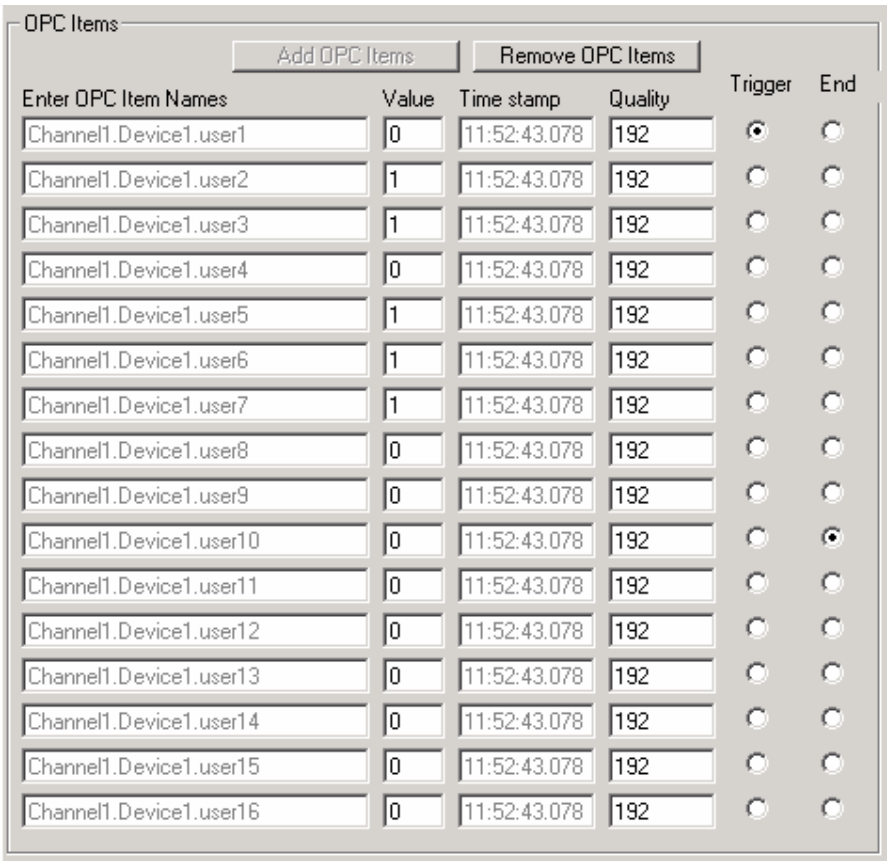


Figure 17: Add Items

4.7.4 Start Recording

After connection is established and OPC items have been added, the start recording button can be pressed to begin the recording process. When the signal marked with the 'Trigger' changes to a high state recording will begin.

Once recording has finished indicated by the progress bar the fields Start, End and Seconds will contain data from the recording process.

Start – Time of Trigger. (start Trigger)

End – Time of end signal.

Seconds – Time elapsed from Trigger to End signal.

Figure 18 shows the function buttons.

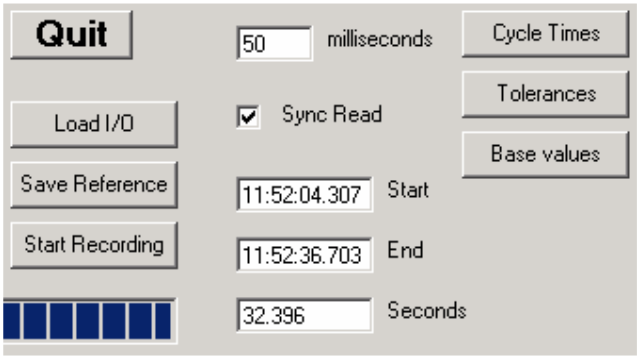


Figure 18: Function buttons

4.7.5 Cycle Time Display

Additionally at the end of recording all signals that have undergone a low – high - low transition will have their sub-cycle times presented in Cycle times section of the display as shown in Figure 19.

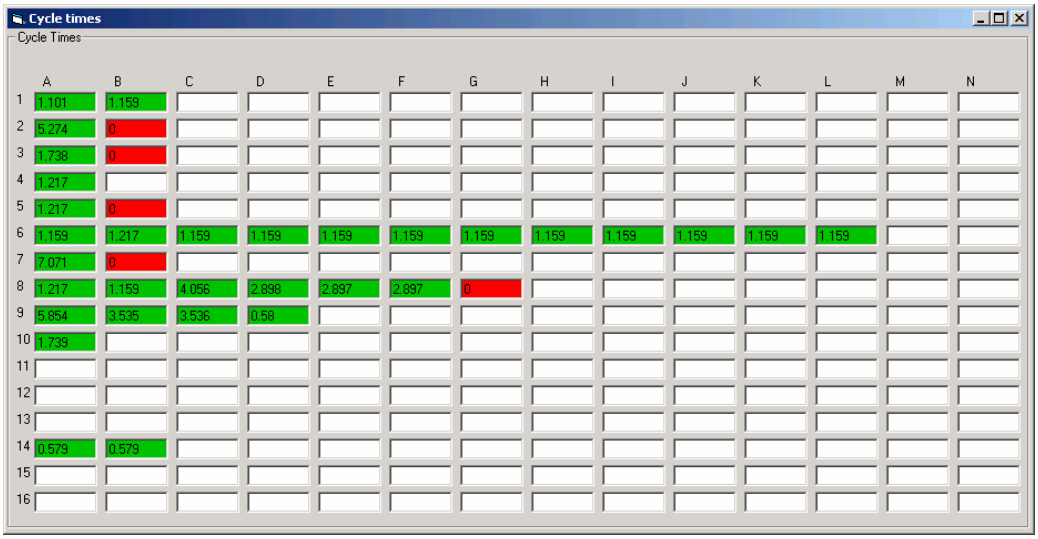


Figure 19: Results display

Times that are displayed in green represent times that fall within the base time +/- the tolerance time.

Times that are displayed in red represent times that fall outside the tolerance times.

Times that are shown in red and have a zero (0) value represent, times that had a high state at the end of the recording cycle.

4.7.6 User Instructions

A comprehensive instruction manual was developed for the cycle time analyser software which can be found in appendix C.

4.8 Testing

Testing involved a number of stages.

1. Testing connection between client and server.
2. Verifying data recording.
3. Report

The first stage was to test the OPC connection and establish that the required information could be accessed by the Visual Basic client program. Initially an attempt was made to establish communication using the corporate network. This involved a number of challenges due to the existing network configuration. With the mixture of domains, workgroups, password rules, access control lists and different subnets, standard DCOM settings could not be adopted to allow communication between the client and server computers. An alternative solution is discussed in section 4.11.2 but as this was not the main focus of this project it was decided to use a computer connected to the same subnet and using the same username / password combination as the server computer for testing purposes.

The next stage of testing involved running the cycle time analyser application and recording the timestamp values for the OPC items selected. This test was first

performed on a piece of equipment that had no moving components to reduce the risk of equipment damage that may have occurred from potential software issues. The test was performed initially by changing the machine operating mode from automatic to manual and recording the associated OPC items. This test proved that accessing the OPC server with a second client had no adverse effects on the systems operation.

The final stage of testing involved taking a number of recordings at different cycle times and comparing the results. A test was conducted by adjusting the robot speed from 100% to 30% and recording the results of the overall cycle time. The results in table 1 show the variation in overall cycle time as the robot speed was varied.

% Speed	Cycle time	Difference
100	36.511	0
75	36.719	0.208
50	38.550	1.831
30	44.428	5.878

Table 2: Overall cycle time variation

4.9 Visualisation of Recorded Data

Microsoft Excel was chosen for the visualisation of the recorded data due to it's ability to easily manipulate and format data. Both data and reference files are stored as tab delimited files and are read into excel using Visual Basic macro's.

Once the data is read into the respective excel sheets Data and Reference, it is formatted to show the data for each of the 16 bits monitored as single step time traces on a displacement diagram as shown in Figure 20.

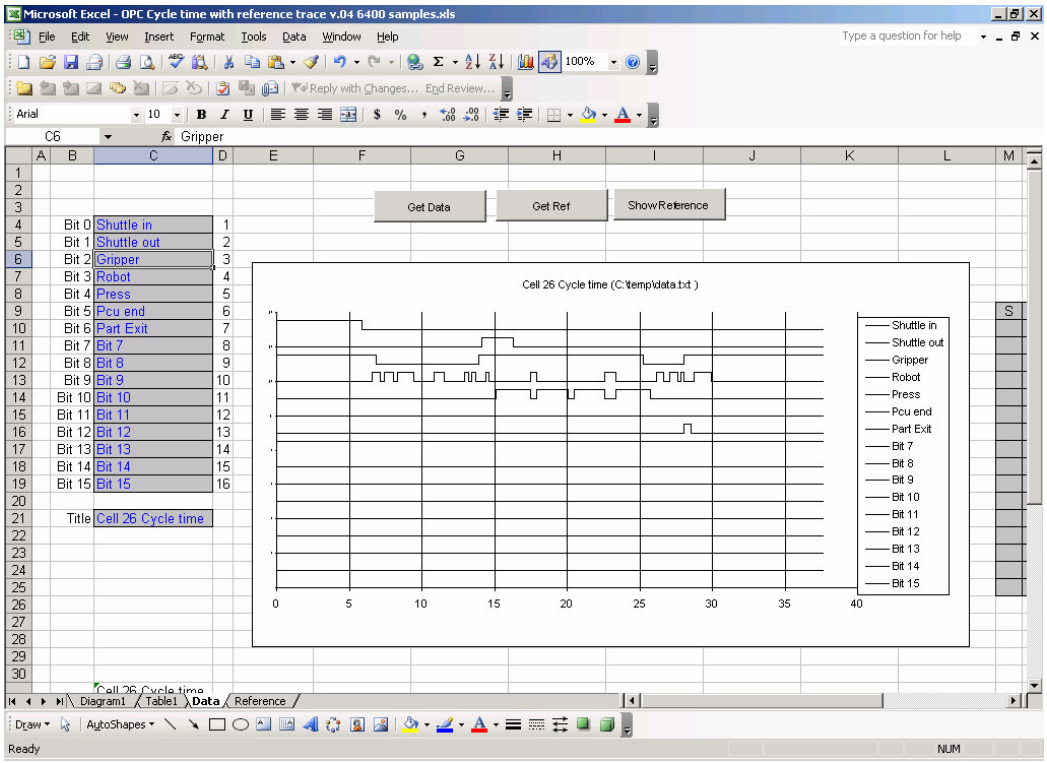


Figure 20: Excel report

The excel report performs the following functions

- Import data and reference files
- Show reference
- Input text description for signals
- Diagram 1, format suitable for printing

4.10 Visual Basic Limitations

The interval property associated with Timer controls in Visual Basic have some inherit limitations.

- If an application is making heavy demands on the system such as long loops, intensive calculations, drive, network, or port access, the cycle time program may not receive timer events as often as the Interval property specifies.
- The system generates 18 clock ticks per second — so even though the Interval property is measured in milliseconds, the true precision of an interval is no more than one-eighteenth of a second.

Considering these limitations it is recommended that this application be run in isolation to reduce the load on the system.

4.11 Problems Encountered

4.11.1 OPC TimeStamp

The majority of OPC servers provide Item timestamp information in UTC format which provides a time resolution down to milliseconds. However Microsoft Visual Basic does not provide any formatting instructions for retrieving milliseconds from the UTC format.

After much investigation and a recommendation from the OPC Foundation Forum the VariantTimeToSystem function was perceived as a solution to this issue.

The VariantTimeToSystemTime function was tested and was also not able to return the millisecond component of the timestamp.

After further investigation, a Microsoft knowledge base 297463 was discovered that supported the conclusion that the VariantTimeToSystemTime was incapable of returning millisecond information.

Finally the decision was made to write a small subroutine to calculate the millisecond value from the timestamp. The milliseconds are then concatenated on to the "hh:mm:ss" format to provide a timestamp with millisecond resolution "hh:mm:ss.000"

Figure 21 shows the code used for the millisecond conversion.

```
' This function converts.UTC based timestamp and extracts milliseconds
‘ Calculate total milliseconds in time stamp
' Convert date time format to double
Dummy1 = CDbl(varDateTime)
' Remove fractional part of number
Dummy2 = Fix(varDateTime)
Dummy3 = varDateTime - Dummy2
Dummy3 = Dummy3 * 100000000
Dummy3 = Int(Dummy3)
TotalmSec = Dummy3
' Extract hours from time stamp
Hours = Fix(Int(Dummy3 / 3600000))
Dummy3 = TotalmSec
' Extract Minutes from time stamp
Minutes = Fix(((TotalmSec - (Hours * 3600000)) / 60000))
Dummy3 = TotalmSec
' Extract seconds from time stamp
Seconds = Fix(((TotalmSec - (Hours * 3600000) - (Minutes * 60000)) /
1000))
' Extract milliseconds from time stamp
Milliseconds = Fix((TotalmSec - (Hours * 3600000) - (Minutes * 60000) -
(Seconds * 1000)))
' Format time output into “00:00:00.000”
GetMilliseconds = Format$(Hours, "00") & ":" & Format$(Minutes, "00") _
& ":" & Format$(Seconds, "00") & "." & Format$(Milliseconds, "000")
```

Figure 21: Code for Millisecond conversion

4.11.2 DCOM Security Settings

Initially an attempt was made to connect to the OPC server from an office computer that was a member of a domain whilst the computer with the OPC server was situated in a workgroup. The configuration required to enable communication using DCOM proved increasingly difficult and was abandoned as it was not the primary

focus of this project. Other alternatives were investigated resulting in the discovery of an OPC tunneller software from a company called Matrikon which effectively connects OPC client and servers independent of DCOM settings. The cost of the OPC tunneller software provided by Matrikon prohibited its use in this project, see Appendix E. Use was made of the 30 day trial period offered by Matikon for evaluation purposes. This software made connection to OPC servers on different subnets remarkably easy and would be considered as a permanent solution to resolve DCOM issues if the cycle time analyser concept is implemented.

4.11.3 SyncRead does not Return Timestamp Data as Array

The SyncRead function has two optional variables, Qualities and Timestamps.

In the OPC Data Access Automation Interface Standard these are defined as "Variants containing a Date array of UTC time stamps for the Timestamps variable and a Variant containing an Integer array of Qualities for the Qualities variable".

Upon further investigation a similar query had already been posted on the OPC foundation's forum page on 14th March 2003, suggesting that the following special logic be used to convert the variants to arrays before the data can be accessed.

```
// Convert Variant containing Date array to Array
Dim TimeStamps() As Double
ReDim TimeStamps(ItemCount)

If VarType(TimeStamp) = vbArray + vbDate Then
Dim Buffer1() As Date
Buffer1 = TimeStamp

   For ii = 1 To ItemCount
      TimeStamps(ii) = CDbl(Buffer1(ii))
   Next ii
End If
```

4.11.4 OPC Server Capability

The basic update rate of the OPC-Servers used for communication is specified as milliseconds = cycle time with which all item data are read from the controller. This data is then written into the cache with which the client communicates with a separately defined update rate.

In comparison to direct access to the controller, reading and writing of variables via this cache list leads to an increase in access time (max. approx. 1ms per item).
The user needs to be aware that using an OPC server with an excessive number of items can reduce the accuracy of the information received from the server.

4.12 Chapter Summary

The Cycle Time Analyser application that has been developed is based on a Windows based application that performs synchronous read requests from the connected OPC server. The collected data represents the 16 OPC items entered in the GUI and is compared to preset values entered into the base times screen and tolerance time screens. The data recorded can be imported into an Excel spread sheet for printing and reporting purposes.

Chapter 5 Conclusions

5.1 Achievement of Objectives

The aim of this project was to develop a software application that was capable of monitoring equipment cycle time utilising the existing infrastructure. The system was developed to the stage where equipment sub cycle times could be extracted from the recorded data and compared with preset values determined by statistical analysis. The software has proven to meet the requirements of a flexible and scalable system suitable for implementation.

The objectives defined in Chapter 1 have been addressed and research into the suitability of OPC and Visual Basic as components of the cycle time analyser have proven to be effective.

The use of existing infrastructure caused a number of authorisation issues which were solved by the use of a third party software application which allows PC connectivity across conflicting network configurations.

The cycle time software was designed and implemented. The system allows real time data to be recorded from equipment and stored in a local file for further analysis. A comparison of recorded data is also performed with pre recorded limits to give an immediate status of equipment cycle time.

5.2 Future Work

The evolution of the cycle time analyser software to it's current level has proven the underlying principles and indicated that the continuation of the development of application would provide an effective monitoring tool. The following tasks are currently seen as enhancements that could be added to the existing solution.

5.2.1 Statistical Evaluation of Tolerance Times

The stability of equipment cycle times is a necessary prerequisite to allow for informed decisions to be made about equipment capability. The calculation of stability and capability were performed by manual calculations as the timing of this project did not allow for this function to be fully integrated into the software application. The implementation of the following functions would greatly increase the speed at which the software could be adapted to different equipment.

- Record 100 samples
- Calculate Capability index
- Calculate lower and upper tolerance limits

5.2.2 Monitor Multiple Machines

The current application only allows for one machine to be monitored at a time. The ability to monitor multiple equipment simultaneously would greatly increase the usability of this application. Processor loading and communication bandwidth will need to be verified to ensure a realistic sampling time is maintained.

5.3 Final Summary

As a result of undertaking this project it has become clear that monitoring equipment cycle time and comparing the values obtained with previously recorded values can be used to determine equipment stability. The software application presented in this project can be used as part of a lean system of operation and used to predict equipment failures. It is anticipated that the cycle time analyser software identified in this project will, after further development and testing, be adopted as a permanent monitoring tool.

References

ISO/TS16949 Technical Specification, Quality Management Systems – *Particular requirements for the application of ISO 9001:2000 for automotive production and relevant service part organizations*

Hobbs, Dennis P, *Lean Manufacturing Implementation: A Complete Execution Manual for Any Size Manufacturer,* J. Ross Publishing

Bill Carreira, 2004, *Lean Manufacturing That Works: Powerful Tools for Dramatically Reducing Waste and maximizing Profits,* Amacom, New York

Toyota Production System Manual, June 1992, Toyota

Murphy E, 2006 'Time for a health check', *Focus: Open connectivity*, 24th December 2006.

Chisholm, A, 1998, 'DCOM, OPC and performance issues', *OPC Foundation white paper*, 1998.

Liu, J, Wee Lim,K, Khuen Ho, W, Chen Tan, K, Tay, A, Srinivasan, R 2005 'Using the OPC Standard for Real-Time Process Monitoring and Control',
IEEE Software, November/December 2005, page 54.

Yoh Shimanuki, 1999, 'OLE for Process Control (OPC) for New Industrial Automation Systems', IEEE

Robert Bosch gmbh
Machine and Process Capability, 3rd edition 2004

Kondor, R, 2007, Matrikon, 'Understanding OPC: Basics for New Users'
http://ethernet.industrial-networking.com/articles/article

How to cycle time
http://www.optimaldesign.com/OLHelp/HowTo/HowToCycleTime.htm

Line Balancing in the Real World, Emanuel Falkenauer, 2007 viewed on 6th May 2007.
http://optimaldsign/Download/OptiLineFalkenauerPLM05.pdf

Birth of the Assembly line, 2007 viewed on 6th May 2007.
http://autopopuli.blogspot.com/2006/10/birth-of-assembly-line.html

Wikipedia, 2007, viewed on 6th May 2007.
http://en.wikipedia.org/wiki/Henry_Ford

Reh, J, 2007 'Benchmarking', viewed on 16th May 2007,
http://management.about.com/cs/benchmarking/a/Benchmarking.htm

Appendix A

Project Specification

A.1 Issue A, 20th March 2007.

University of Southern Queensland
Faculty of Engineering and Surveying

ENG 4111/4112 Research Project
PROJECT SPECIFICATION

For:	Clement Bollaart
Topic:	Equipment cycle time analyser
Supervisor:	Dr Paul Wen
Enrolment:	ENG4111 – S1, X, 2007 ENG4112 – S2, X, 2007
Project Aim:	To produce a software package that notifies the user when equipment cycle time deviates from a predefined best case example.

PROGRAMME: Issue A, 20th March 2007.

1. Research possible methods to collect real time data from industrial devices.
2. Analyse complete equipment cycle and divide operations to logical steps to be monitored by cycle time analyser.
3. Design software programme to record step times from online equipment.
   - Record and store best case example.
4. Investigate and define tolerance times allowed for various steps (cylinder movements, barcode reading, screwing operations, vision inspection etc)
5. Extend software to check real time data collected from a machine cycle with pre-defined times allowed for steps.
   - User to be notified when current equipment cycle deviates + or - form desired best case example.
   - Step which caused cycle time deviation + or - to be identified.
6. Evaluate deviations from best case example, and use data collected to maintain and improve equipment reliability.
7. Investigate the potential use of such a monitoring device as a predictive maintenance tool to continually improve the effectiveness and efficiency of production equipment, as required by ISO/TS16949 Particular requirements for the application of ISO 9001:2000 for automotive production and relevant service part organizations.

As time permits

8. Extend application to monitor multiple equipment simultaneously.
9. Extend application to notify user via email if equipment cycle continuously falls out side predefined limits.

AGREED: C L Bollaart (Student) (Paul) (Supervisor)

Dated 26/03/2007 Dated 4/4/2007

Approved AOD 14/4/07

Appendix B

OPC Input Items Example

```
VAR_CONFIG
(*   Author:    PlcConfigurator (V3.7 from 19.05.2005)
     ConfDate:  22.08.2006
     Revision:  xxRev
     Who?:      xxWho
     Date:      xxDate         *)
(*---------------------------------------------------------------------
   Variable declaration of the I/O's for every object in this station.
  ---------------------------------------------------------------------*)
(*  A380: Robot (STAEUBLITCPIP) *)
   .Robot01.RobAxisReady   AT %IX22.0   :BOOL;
   .Robot01.RobHighPower   AT %IX22.1   :BOOL;
(*  M940Z: Gripper (Valve) *)
   .M940Z.AInitPosition   AT %IX108.5   :BOOL;
   .M940Z.AWorkPosition     :BOOL := FALSE;
   .M940Z.BInitPosition     :BOOL := FALSE;
   .M940Z.BWorkPosition     :BOOL := FALSE;
   .M940Z.CoilA   AT %QX20.1   :BOOL;
   .M940Z.CoilB   AT %QX20.2   :BOOL;
(*  ESP: GripperToolCheck (SensorCheck) *)
   .ESP.SensorToCheck   AT %IX108.7   :BOOL;
(*  A387: Press (PCU1000) *)
   .Press01.ReadyToWork   AT %IX50.0   :BOOL;
   .Press01.ProgOutBit0   AT %IX50.1   :BOOL;
   .Press01.ProgOutBit1   AT %IX50.2   :BOOL;
   .Press01.ProgOutBit2   AT %IX50.3   :BOOL;
   .Press01.ProgOutBit3   AT %IX50.4   :BOOL;
   .Press01.ProgOutBit4   AT %IX50.5   :BOOL;
   .Press01.ReadyToStart   AT %IX50.6   :BOOL;
   .Press01.GenError   AT %IX50.7   :BOOL;
   .Press01.CycleFin   AT %IX51.0   :BOOL;
   .Press01.NetworkError   AT %IX51.1   :BOOL;
   .Press01.ShutdownOK   AT %IX51.2   :BOOL;
   .Press01.EmerStop   AT %QX50.0   :BOOL;
   .Press01.ProgInBit0   AT %QX50.1   :BOOL;
   .Press01.ProgInBit1   AT %QX50.2   :BOOL;
   .Press01.ProgInBit2   AT %QX50.3   :BOOL;
   .Press01.ProgInBit3   AT %QX50.4   :BOOL;
   .Press01.ProgInBit4   AT %QX50.5   :BOOL;
```

Appendix C

Cycle Time Analyser Instruction Manual

Control 2000 RBAU

Cycle Time Analyser

System for Analysis of Cycle Time
of Production Equipment
with
OPC

Instruction Manual

BOSCH

OPC Cycle time analyser3.doc

RBAU/MFA1, 09/2007

Control 2000 RBAU

Cycle Time Analyser

1. Table of Contents

1. Table of Contents 57
2. Purpose and Aim of Cycle Time Analysis 58
   2.1. Extension of cycle time monitoring 58
3. System Configuration 59
4. Summary 59
5. Prerequisites 59
6. Program Installation 60
7. Network setup 60
8. DCOM settings 60
9. Cycle Time Analyser Operation 61
   - 9.1 Loading Items from File 62
   - 9.2 Connect to OPC Server 63
   - 9.3 Add Group to OPC Server 64
   - 9.4 Adding and Displaying Items 64
   - 9.5 Start recording 66
   - 9.6 Cycle time results 67
   - 9.7 Base times 68
   - 9.8 Tolerance times 69
   - 9.9 Displaying results in excel 70
     - 9.9.1 Input sheet data 72
     - 9.9.2 Zoom 73
     - 9.9.3 Excel Diagram 1 74
10. Change index 75

2. Purpose and Aim of Cycle Time Analysis

Frequently the requirement exists to optimise the cycle time of existing production, assembly or test equipment.

In the case of complex control systems, the chronological reference to the state of sensors and other components are not directly obvious from the machine program.

Deviations from the designed cycle time are difficult to locate.

The method applied so far is to record signals using a multi-channel recorder.

This method has a number of disadvantages:

- Only possible to monitor sensors or hard signals.
- Recording of a very limited number of signals.
- Considerable effort is required to connect individual signals.

The aim is to generate an analyser diagram from the control system of the equipment.

With this system of cycle time analysis it is possible to chronologically record all signal changes of the selected inputs / outputs of a complete machine cycle.

The system is connected to the automation controller.

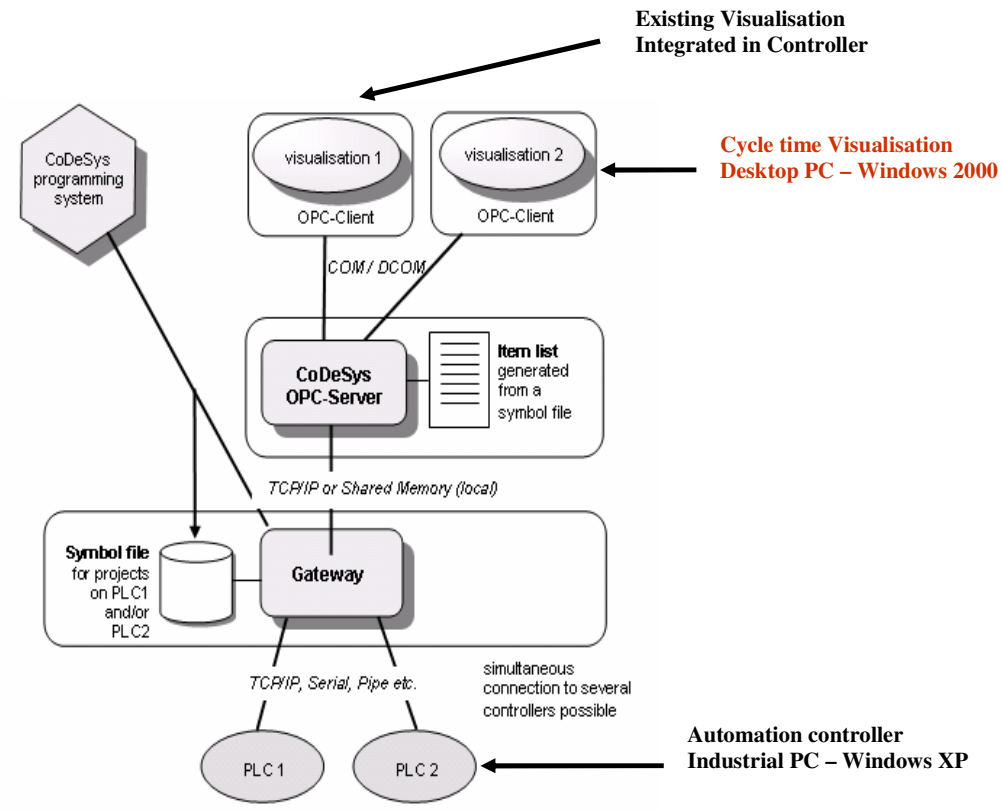
2.1. Extension of cycle time monitoring

By specifying a base value and allowable tolerance limit the complete sequence can be monitored.

Control 2000 RBAU

Cycle Time Analyser

3. System Configuration



4. Summary

Prior to recording, the machine specific inputs and outputs are allocated to items and entered in the item allocation table. There is a facility to read the input items from a file.
After recording, data is stored in a text file “data.text" and can be imported to the cycle time analyser Excel sheet for visualisation and analysis.

5. Prerequisites

Microsoft Excel
Access to Equipment subnet
PC with required DCOM and network configuration.

6. Program Installation

The PC program is available on disk as “OPC Cycle Time Analyser.exe” and may be copied to any directory.
The recording sample is stored in the temp directory on C drive: “C:\temp\data.text
A reference sample can also be saved on C drive “C:\temp\ref.

7. Network setup

This section describes how to adapt the network configuration to allow your PC to connect using TCPIP with the equipment to be analyzed.

- Change DCOM settings on your PC and equipment to those recommended in section 8
- Connect your PC to same subnet as the installed equipment.
- Change your IP address to one that is located in the same subnet as your equipment
- Create a user with the same login details “user name and password” as the target equipment.
- Map a network drive to the target equipment to be monitored.

8. DCOM settings

See “Windows XP SP1 DCOM Configuration” for details on how to change your computers DCOM settings to allow the cycle time analyser software to communicate with your machine.

Control 2000 RBAU

Cycle Time Analyser

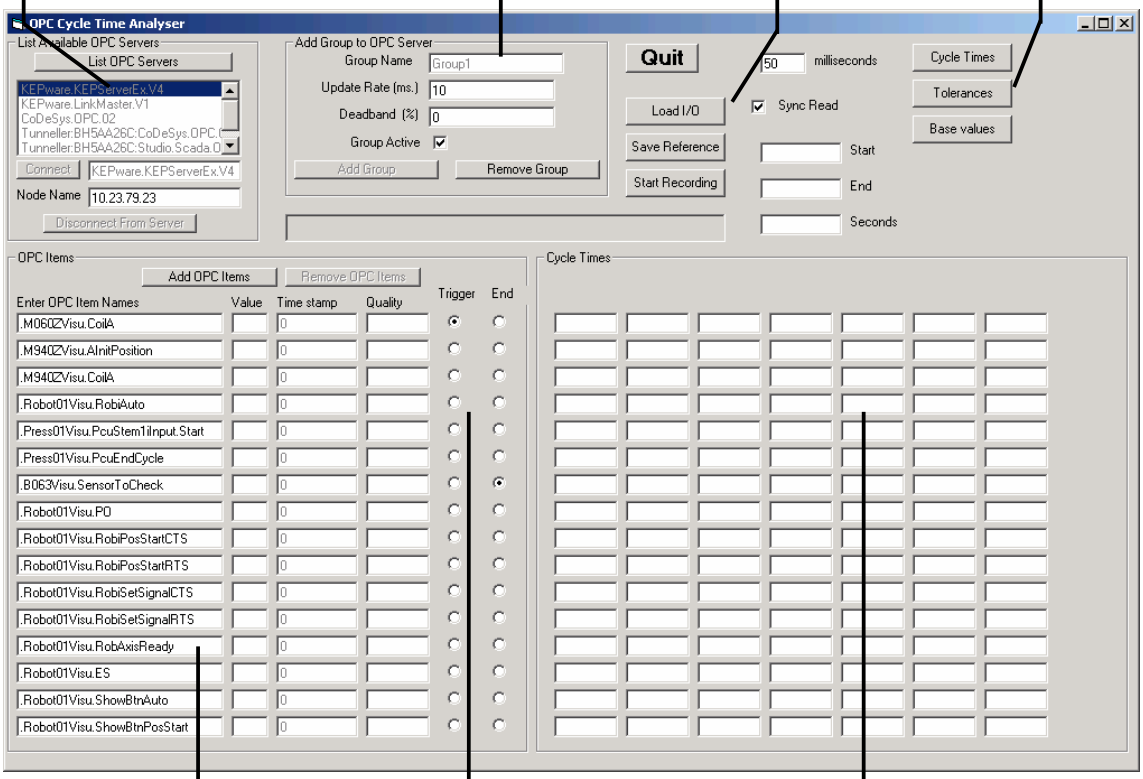
9. Cycle Time Analyser Operation

OPC Server Connection:
Connection is established to required OPC server by listing all servers available and selecting required server

OPC Group:
Enter name of OPC server group.

Function Buttons:
Operator button for starting , loading Items and saving reference diagram.

Screens:
These buttons open the cycle times, Tolerance and base values screens



OPC Items:
16 items can be monitored by the OPC Analyser and are entered into these fields. Alternatively Items can be loaded from a pre written text file with the Load I/O function.

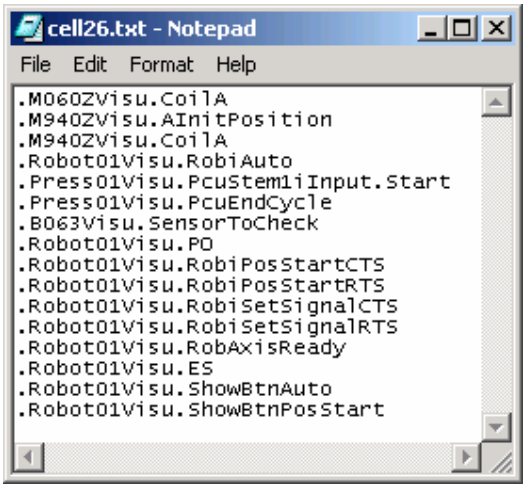
Triggers:
Triggers provide a means to start and end measuring on individual Items. The total time lapsed from start to end is displayed top left of screen.

Cycle times:
Measured cycle times are displayed after the measurement time is complete.

9.1. Loading Items from File

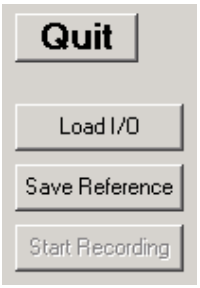
The list of OPC Items to be monitored can also be loaded from a text file. This makes the adaptation for another machine much quicker than entering individual items.

The file must contain the proper OPC item descriptions, similar to those in the example below.



Enter a maximum of 16 items to be copies to OPC Items.
This file can be created in a text editor like Notepad or Ultraedit.

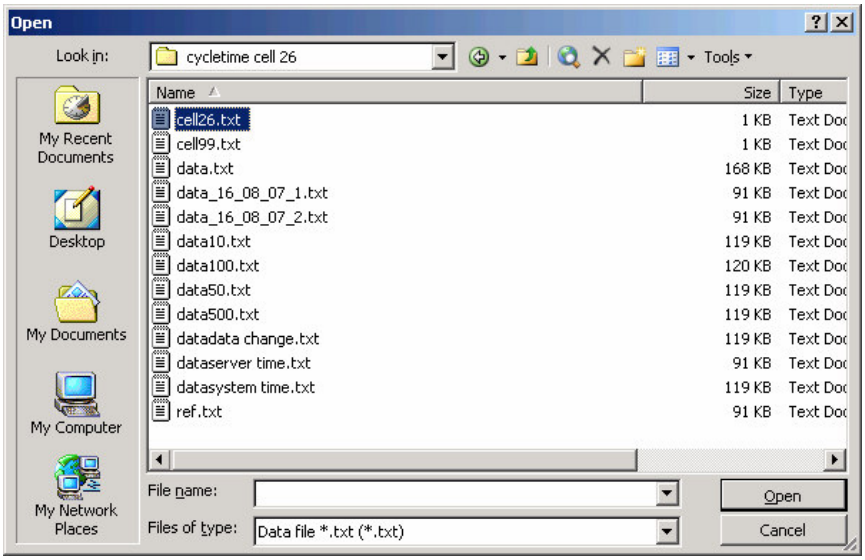
Click on the Load I/O to open the I/O text file →



Select the appropriate file. Then click Open

Control 2000 RBAU

Cycle Time Analyser

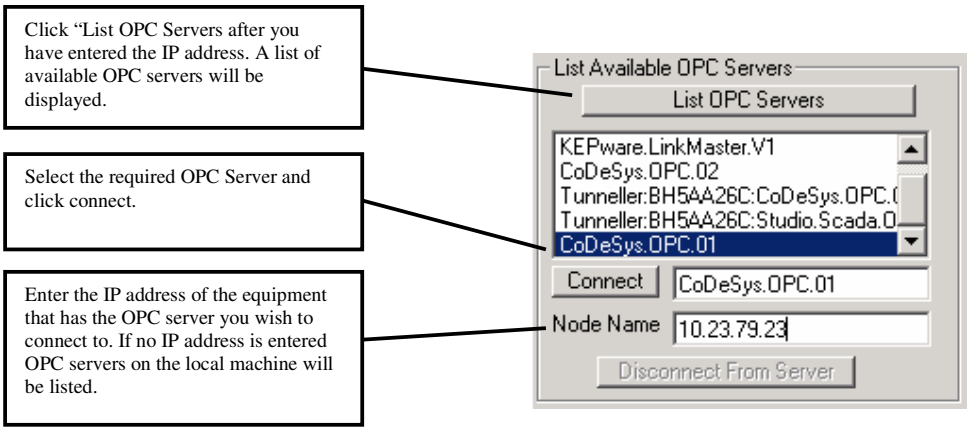


OPC Items are now copied from the I/O file to the OPC Items in the cycle time analyser screen.

9.2. Connect to OPC Server

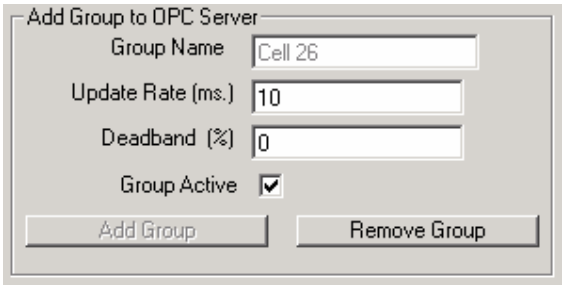
This section describes how to list and connect to an OPC server:

- Input Node Name IP address
- Click ***List OPC Servers***
- Select the required OPC server
- Click ***Connect***



9.3. Add Group to OPC Server

This section describes how to add a group to the OPC server
- Type the Group Name
- Click ***Add Group***

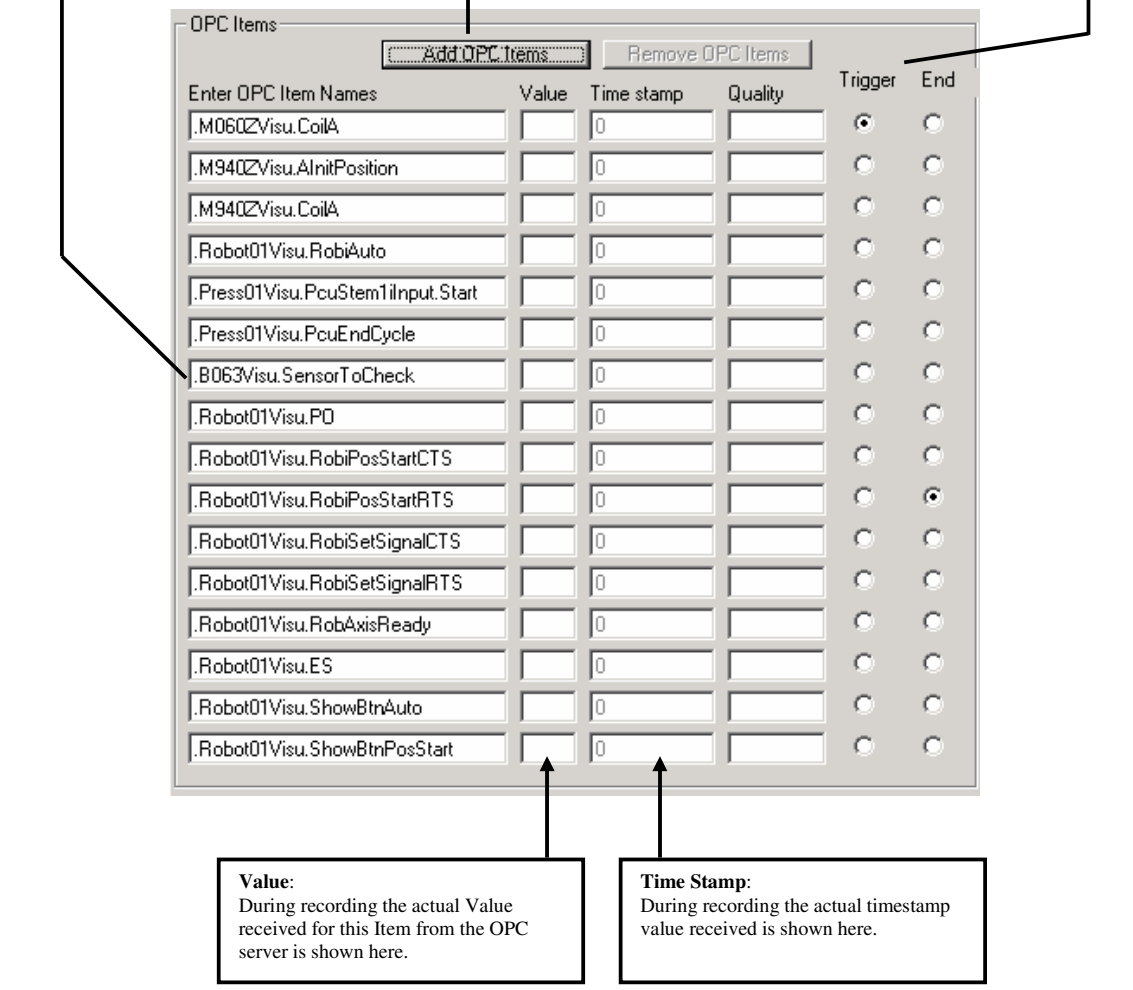


9.4. Adding and Displaying Items

This section describes how to add OPC items.
- Type the OPC Items or load OPC Items as described in Section 9.1.
- Select the required Trigger and End signals.
- Click Add OPC Items.

Control 2000 RBAU

Cycle Time Analyser



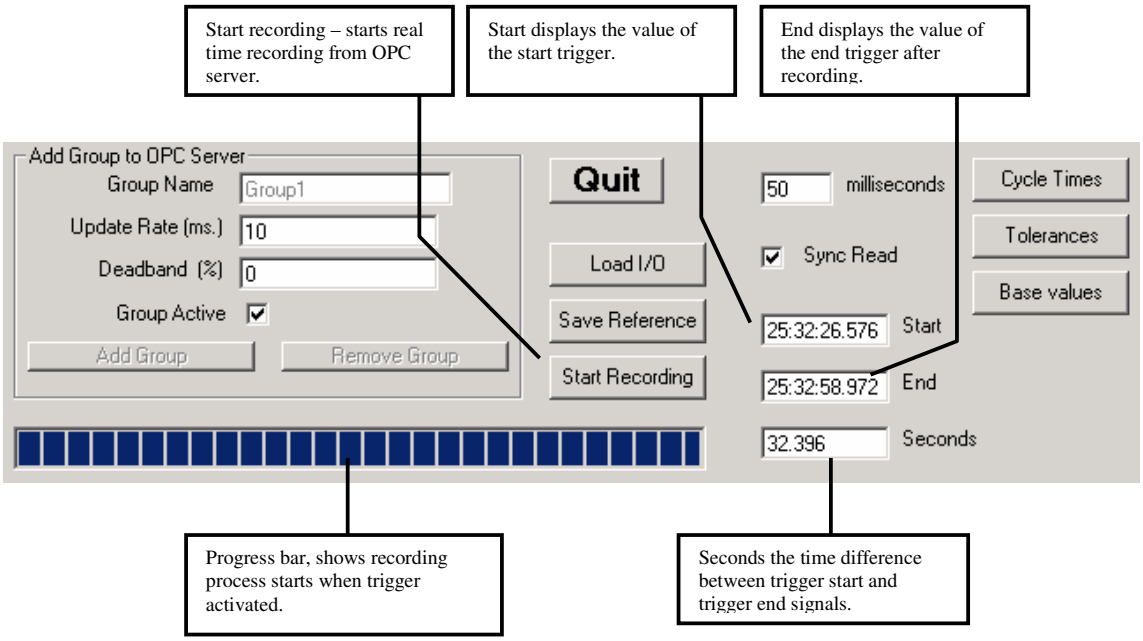
Control 2000 RBAU

Cycle Time Analyser

9.5. Start recording

This section describes how the analyzer is started and some other related functions.

- Click start recording button.
- Time stamp values should now be changing as items are read from the server
- Once the selected trigger item has a 0 to 1 transition the recording progress bar will be active.
- The progress bar will show when the recording process is finished.
- Once recording is finished the fields Start, End and Seconds will display values:
  Start - is the time the trigger 0 to 1 transition.
  End is the time of the end trigger 0 to 1 transition
  Seconds is the calculated difference



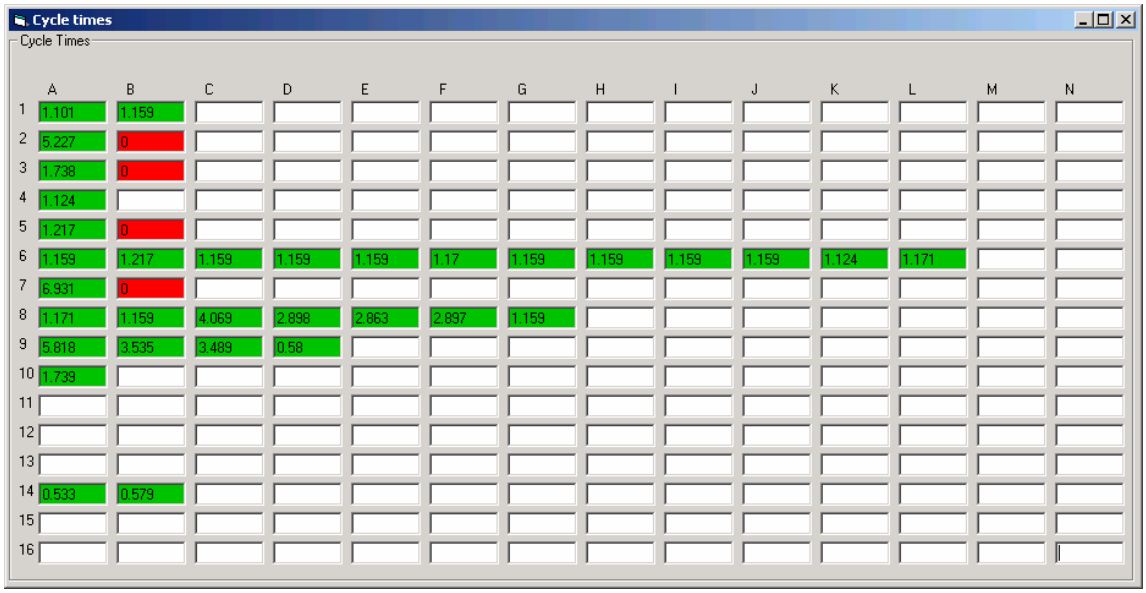
Control 2000 RBAU

Cycle Time Analyser

9.6. Cycle time results

This section shows the Cycle times results screen.

- Items highlighted in green identify items within the tolerance time.
- Items highlighted in red identify items outside the tolerance time.
- Items highlighted in red and with zero (0) value identify items that were still in a high state at the end of recording.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	1.101	1.159												
2	5.227	0												
3	1.738	0												
4	1.124													
5	1.217	0												
6	1.159	1.217	1.159	1.159	1.159	1.17	1.159	1.159	1.159	1.159	1.124	1.171		
7	6.931	0												
8	1.171	1.159	4.069	2.898	2.863	2.897	1.159							
9	5.818	3.535	3.489	0.58										
10	1.739													
11														
12														
13														
14	0.533	0.579												
15														
16														

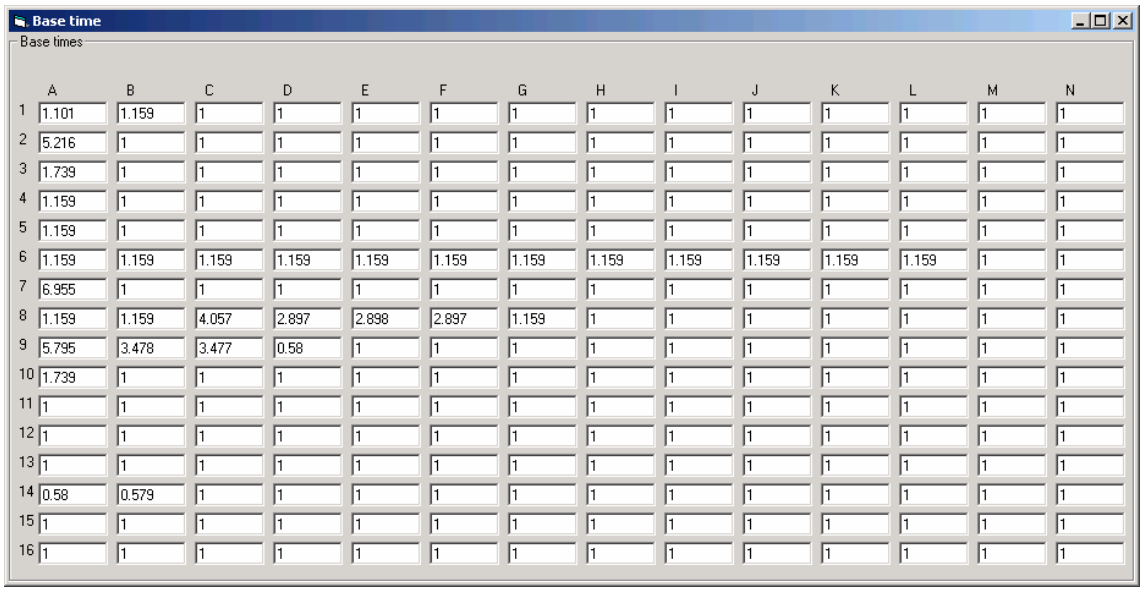
Control 2000 RBAU

Cycle Time Analyser

9.7. Base times

This section shows the Base times screen.

- Base times obtained by statistical analysis of process cycles are entered on this screen. The mean value should be used.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	1.101	1.159	1	1	1	1	1	1	1	1	1	1	1	1
2	5.216	1	1	1	1	1	1	1	1	1	1	1	1	1
3	1.739	1	1	1	1	1	1	1	1	1	1	1	1	1
4	1.159	1	1	1	1	1	1	1	1	1	1	1	1	1
5	1.159	1	1	1	1	1	1	1	1	1	1	1	1	1
6	1.159	1.159	1.159	1.159	1.159	1.159	1.159	1.159	1.159	1.159	1.159	1.159	1	1
7	6.955	1	1	1	1	1	1	1	1	1	1	1	1	1
8	1.159	1.159	4.057	2.897	2.898	2.897	1.159	1	1	1	1	1	1	1
9	5.795	3.478	3.477	0.58	1	1	1	1	1	1	1	1	1	1
10	1.739	1	1	1	1	1	1	1	1	1	1	1	1	1
11	1	1	1	1	1	1	1	1	1	1	1	1	1	1
12	1	1	1	1	1	1	1	1	1	1	1	1	1	1
13	1	1	1	1	1	1	1	1	1	1	1	1	1	1
14	0.58	0.579	1	1	1	1	1	1	1	1	1	1	1	1
15	1	1	1	1	1	1	1	1	1	1	1	1	1	1
16	1	1	1	1	1	1	1	1	1	1	1	1	1	1

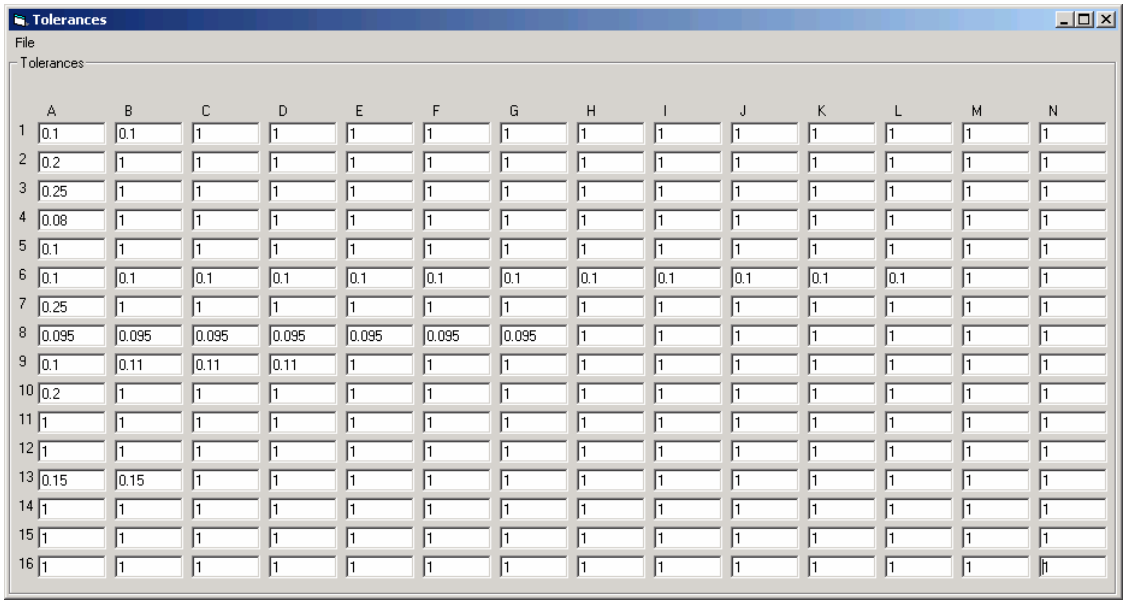
Control 2000 RBAU

Cycle Time Analyser

9.8. Tolerance times

This section describes the Tolerances screen.

- Tolerance times obtained by statistical analysis of process cycles are entered on this screen.
- Tolerance value should be determined by statistical analysis +/- 3s.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	0.1	0.1	1	1	1	1	1	1	1	1	1	1	1	1
2	0.2	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0.25	1	1	1	1	1	1	1	1	1	1	1	1	1
4	0.08	1	1	1	1	1	1	1	1	1	1	1	1	1
5	0.1	1	1	1	1	1	1	1	1	1	1	1	1	1
6	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	1	1
7	0.25	1	1	1	1	1	1	1	1	1	1	1	1	1
8	0.095	0.095	0.095	0.095	0.095	0.095	0.095	1	1	1	1	1	1	1
9	0.1	0.11	0.11	0.11	1	1	1	1	1	1	1	1	1	1
10	0.2	1	1	1	1	1	1	1	1	1	1	1	1	1
11	1	1	1	1	1	1	1	1	1	1	1	1	1	1
12	1	1	1	1	1	1	1	1	1	1	1	1	1	1
13	0.15	0.15	1	1	1	1	1	1	1	1	1	1	1	1
14	1	1	1	1	1	1	1	1	1	1	1	1	1	1
15	1	1	1	1	1	1	1	1	1	1	1	1	1	1
16	1	1	1	1	1	1	1	1	1	1	1	1	1	1

9.9. Displaying results in Excel

This section describes how to view recorded results in Excel.

- Using the Excel spread sheet file provided with the software named “OPC Cycle time” you can view the recorded data as a displacement / time diagram as shown below.
- The spread contains macros which must be enabled for proper operation.
- The spread sheet contains 4 pages **Table1**, **Diagram1**, **Data** and **Reference**.
- Click on the Get Data button and search for your recorded data file. Default location is C:\temp\data.txt.

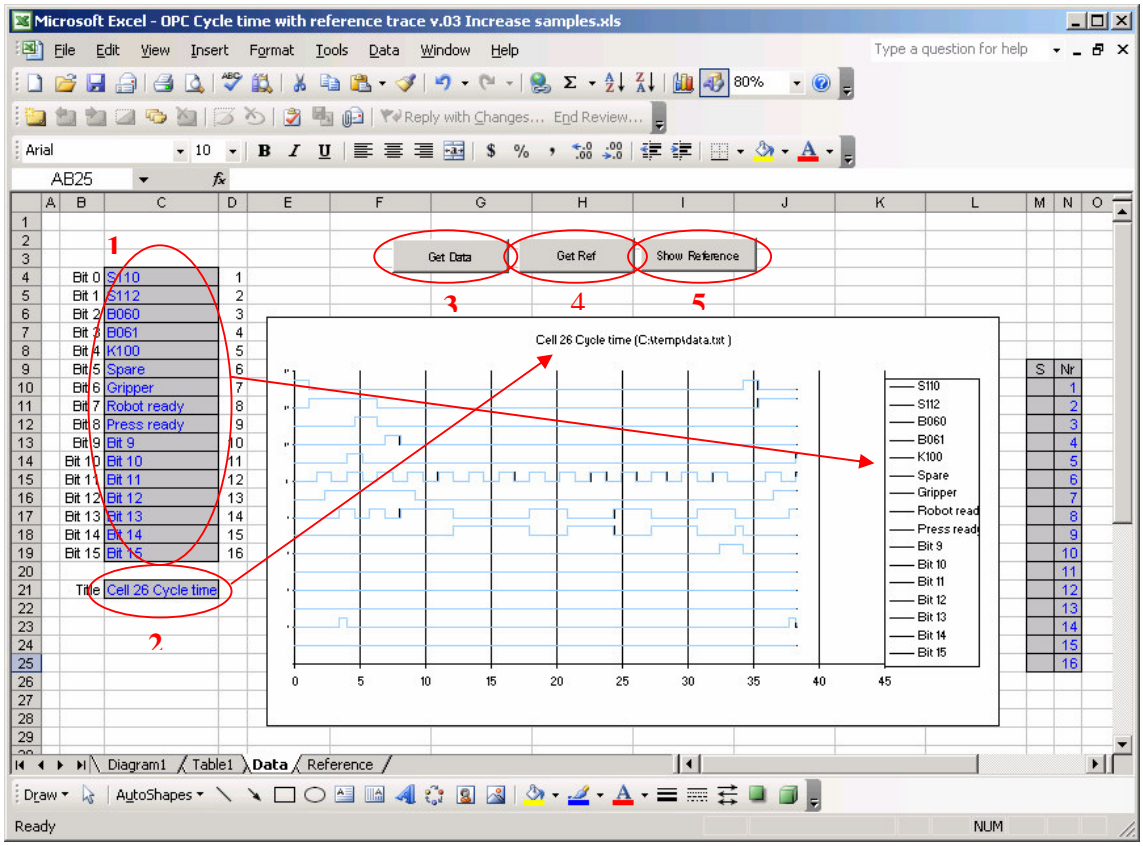
The data loaded into excel is stored and additional values calculated in **Table1** sheet. The reference curve data is stored and additional values calculated in the **Reference** sheet.

Diagram1 is a printable version of the actual cycle time diagram.
See section 9.9.3 for further details.

The most important sheet is the **Data** sheet described below.

Control 2000 RBAU

Cycle Time Analyser

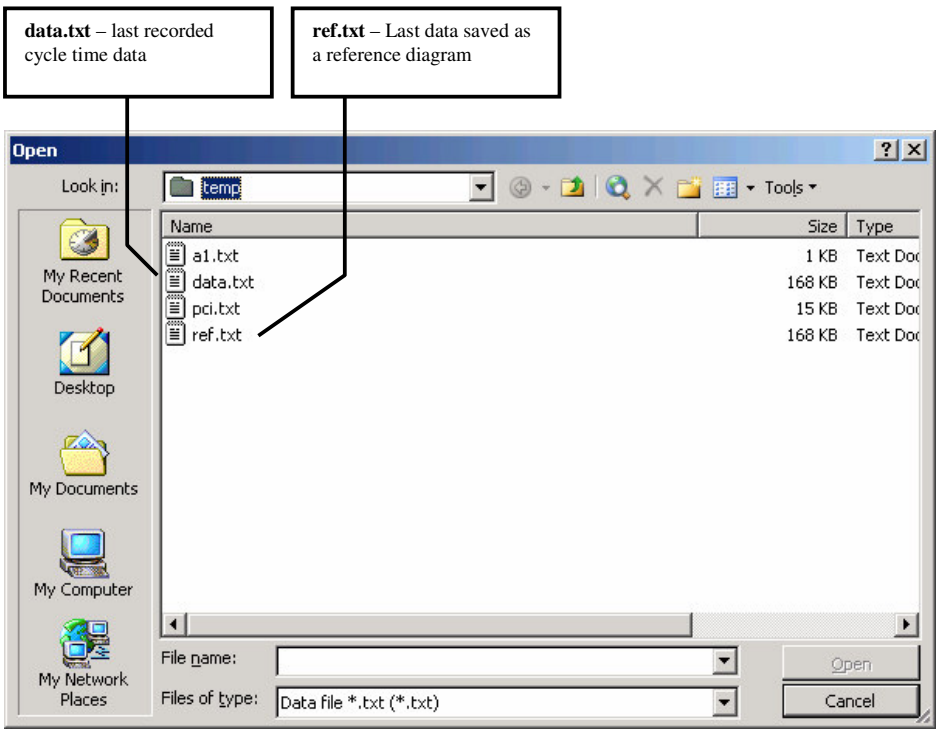


Control 2000 RBAU

Cycle Time Analyser

9.9.1. Input sheet data

The **switches 3** and **4** are used to load cycle time and reference data. They open the following Dialog.



Select the appropriate file data.txt or ref.txt, other files names will cause an error message.

In **area 1** of the excel sheet you can enter a symbolic name. The name entered is transposed to the right of the cycle time diagram to identify the individual signals.

In **area 2** of the cycle time diagram you can enter a name for the diagram. This name also appears as a heading for the cycle time diagram.

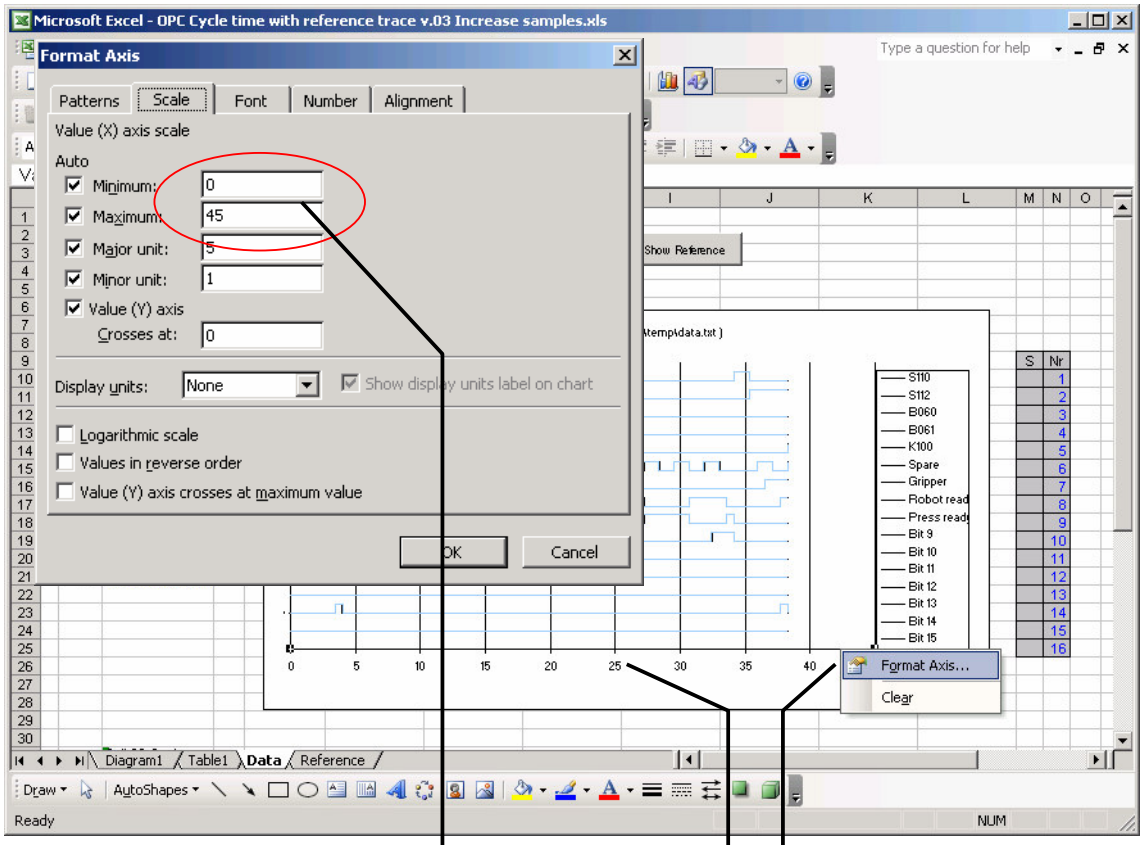
Control 2000 RBAU

Cycle Time Analyser

9.9.2. Zoom

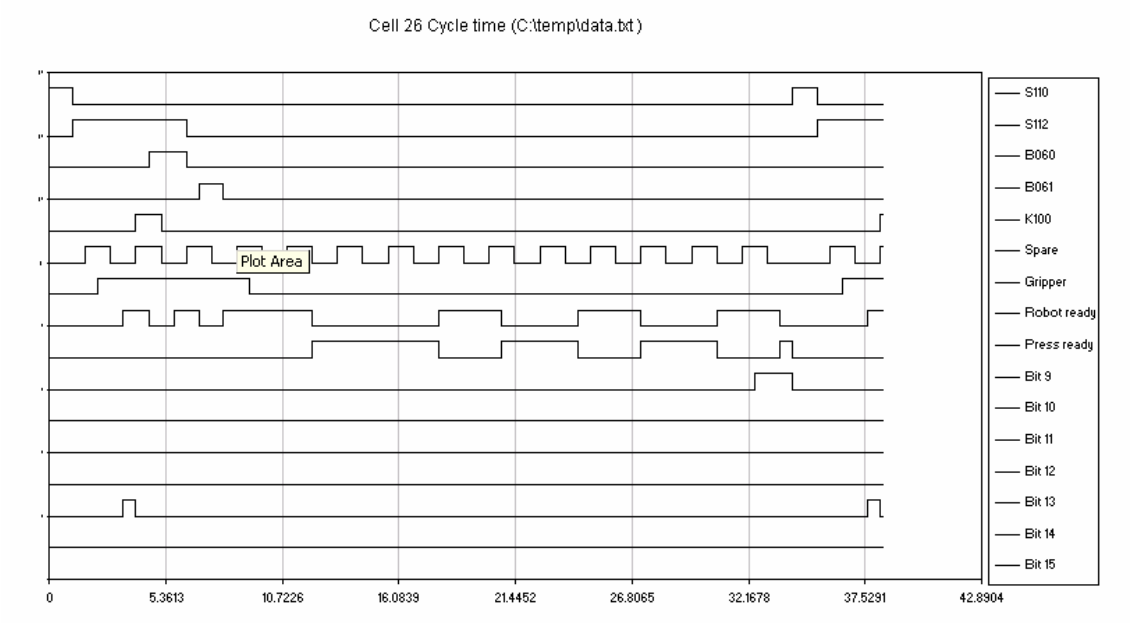
For a better resolution of the cycle time diagram use the Excel function to change the axis scale.
With the right mouse button, click on the numbers of the X axis, then select **Axis Format**, then change the axis scale appropriately.

The minimum and maximum values can be changed to view new start and end points of the cycle time diagram.



9.9.3. Excel Diagram 1

The Excel sheet **Diagram 1** is provided for documentation purposes, it is presented in a format which is better for printing.



Control 2000 RBAU

Cycle Time Analyser

10. Change index

Version	Date	Completed by	Changes
1.0	27.09.07	MFA1 Bollaart	First issue

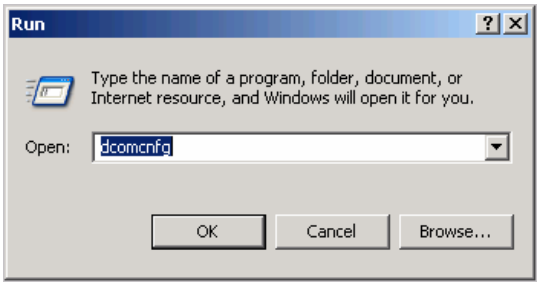
Appendix D

DCOM Setting for windows XP

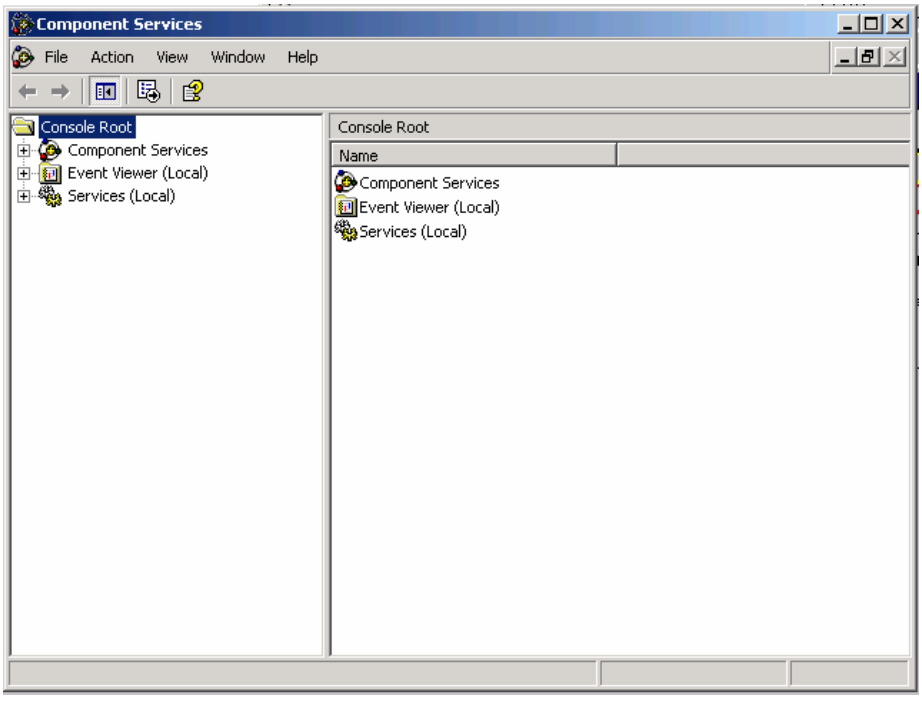
Windows XP DCOM Configuration For OPC Cycle time Analyser

Notes: These directions will open up DCOM to **all** users.
After these steps have been completed and communication has been established it is recommended that DCOM permissions be tightened to only allow the necessary users access.
These setting apply to Windows XP SP1

1. From the start menu go to run and type dcomcnfg



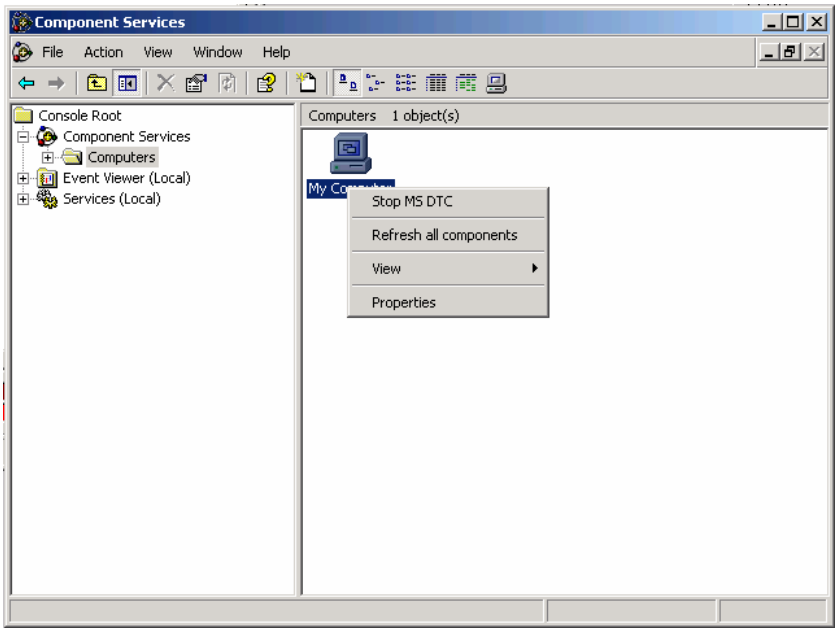
2. Select component services



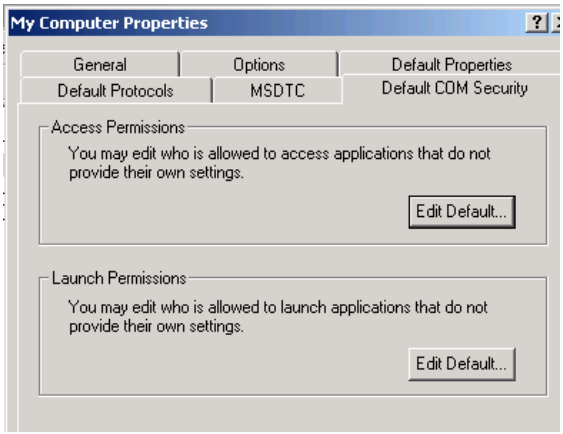
Control 2000 RBAU

DCOM Configuration

3. Under component services right click on My Computer



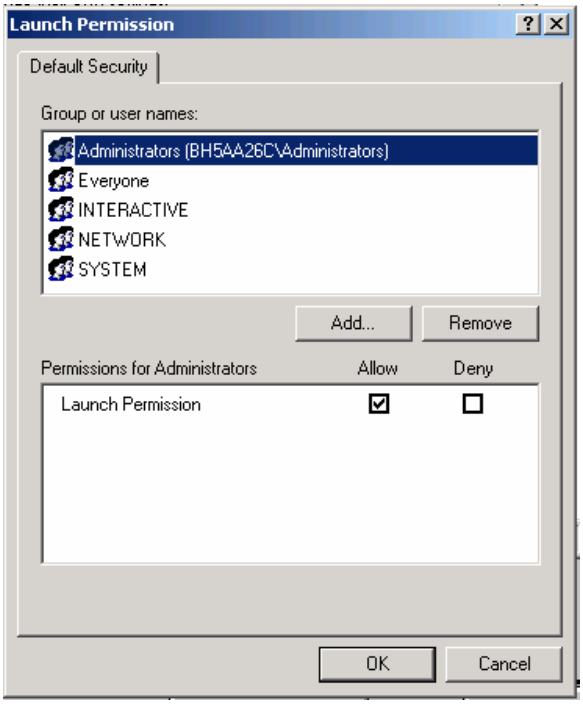
4. Go to the “Default COM Security” tab.

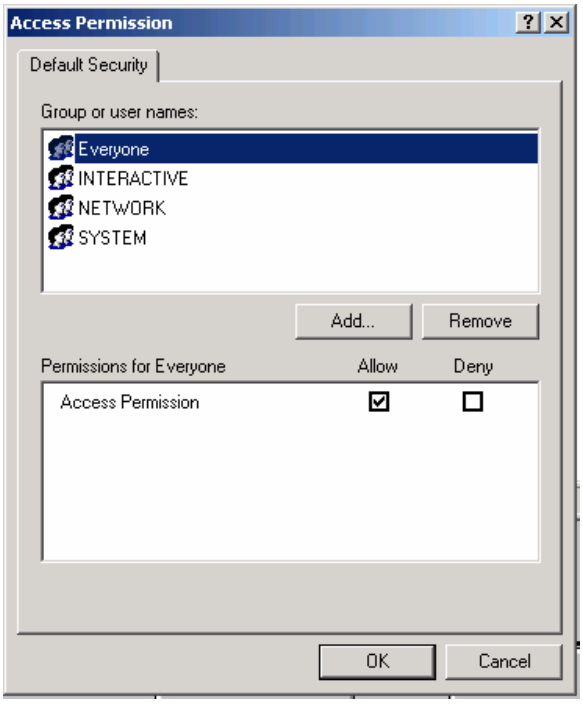


5. Ensure that both Access and Launch Permissions have “Everyone”, “Interactive”, “Network”, and System set to Allow.

Control 2000 RBAU

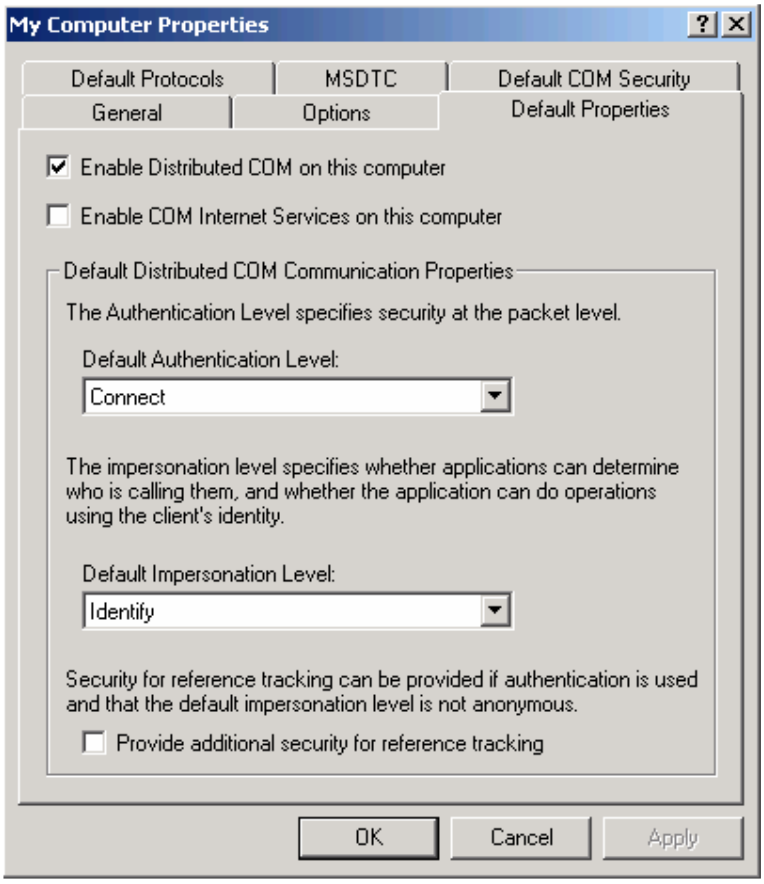
DCOM Configuration





DCOM Configuration

6. Now go back to the “My Computer Properties” dialog. Click on the “Default Properties” tab. Ensure that the settings on the machine match those in the screenshot below.



7. Click OK. You should now be looking at the “Component Services” panel.

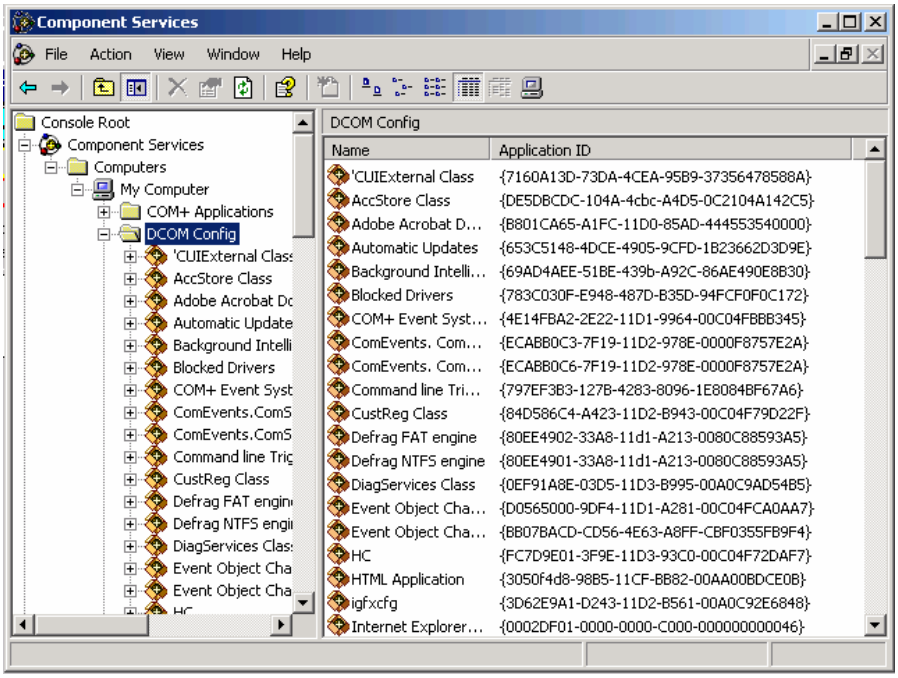
8. Open the “My Computer” tree and then the “DCOM Config” Folder. Inside this folder are all the DCOM Applications installed on the local machine. We need to ensure that all of our OPC servers are using the appropriate permissions.

9.

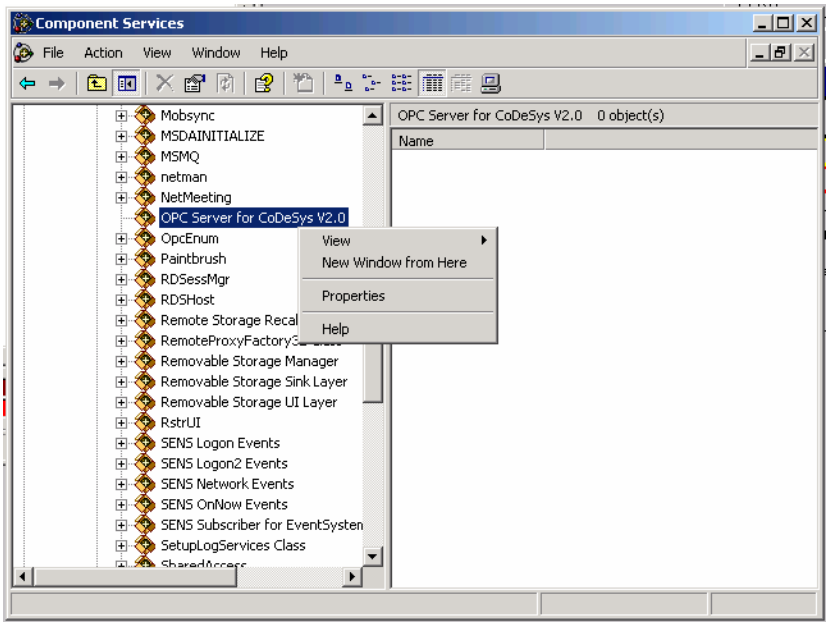
Note: Due to a limitation in the windows XP DCOM Configuration tool, not all OPC servers may be listed as their Application name, but instead as their class ID number. If the OPC server is being listed by it’s class ID.

Control 2000 RBAU

DCOM Configuration



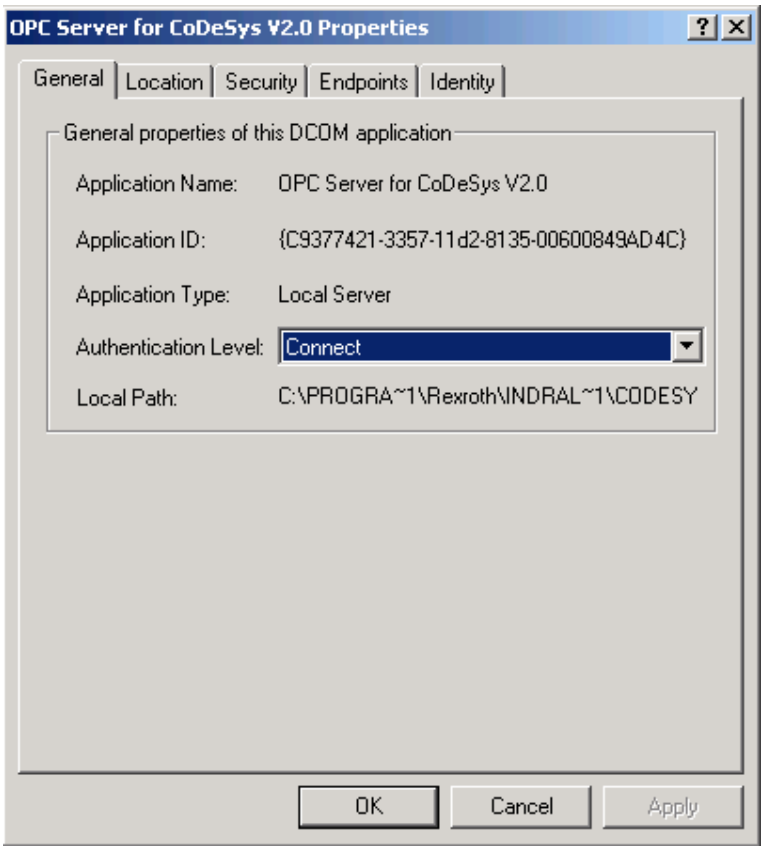
10. Find your OPC Server in the list. Right Click on the OPC Server and select Properties.



Control 2000 RBAU

DCOM Configuration

11. Ensure that your Authentication level is set to “Connect”.



12. If you have successfully set up the defaults for the system, select the server to use the default settings.

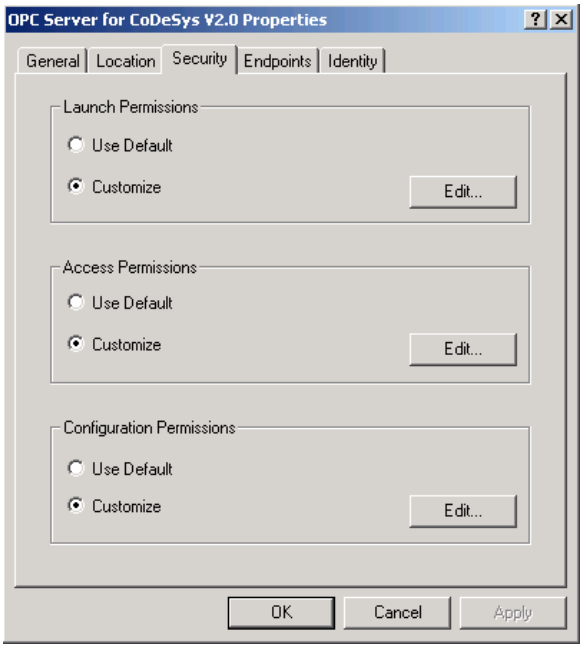
13. Under the “Identity” tab, ensure that the OPC server is running as “The Interactive User”.

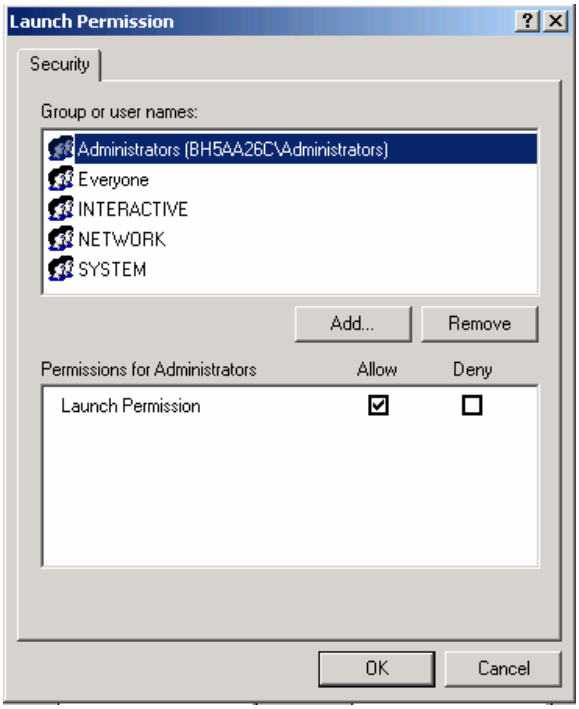
14. Now that customise has been selected, please edit the default launch and access permissions to contain the following users, “Everyone, “Network”, “Interactive”, “System”.

15. Make sure that all of these users are set to allow.

Control 2000 RBAU

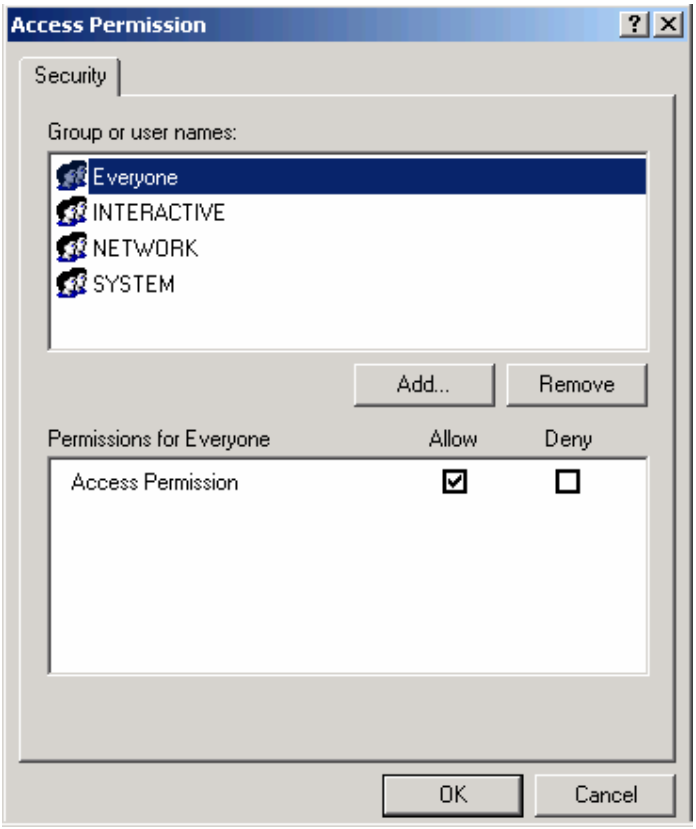
DCOM Configuration





Control 2000 RBAU

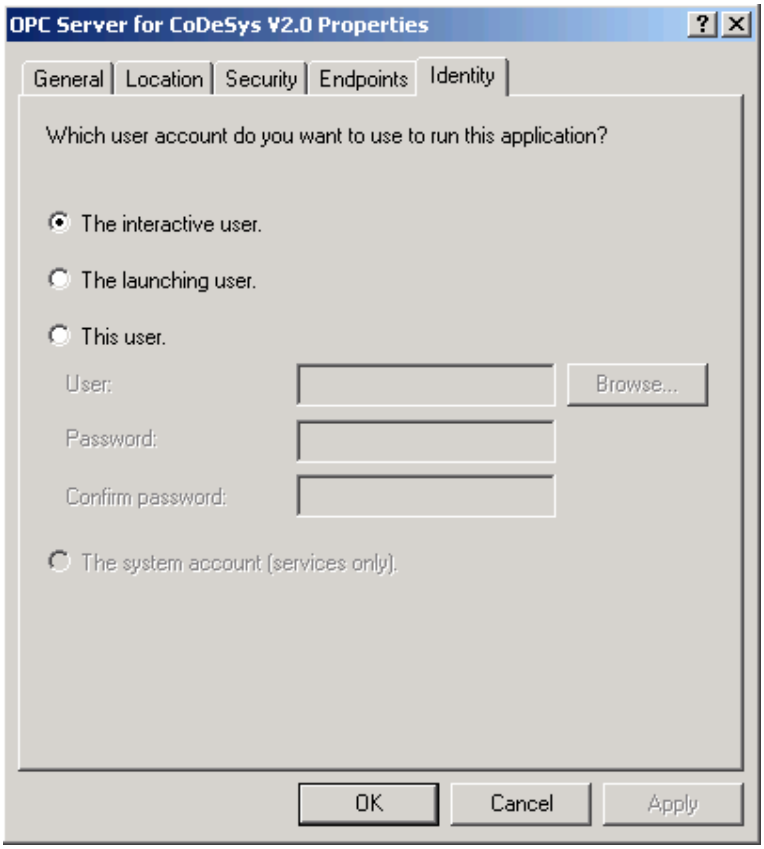
DCOM Configuration



16. Under the Identity tab, please ensure that your OPC Server is running as “The Interactive User”.

Control 2000 RBAU

DCOM Configuration



Appendix E

OPC Tunneller Quotation

From: Bob Erickson
Sent: Wednesday, August 08, 2007 2:09 PM
To: 'Bollaart Clem (CC/MFA1-AU)'
Subject: RE: OPC Tunneller

Hi there Clem,

Good talking to you today about your project using Tunneller to eliminate your need for DCOM configuration.

I'm pleased to provide you with the following quotation:

***** Quotation *****

1 pc. OPC Tunneller (either Client or Server end)-- AU$ 900

***** Suggested Options *****

- Annual Maintenance special (3 additional years for the price of 2) -- AU$ 300
- OPC Troubleshooting Training
- Site Licenses are available for this software

***** Implementation Services *****

As well, we offer onsite implementation services. Therefore, if you are running out of human resources or have better things to do, we can send one of our people to complete the install for you. This would make the entire operation turn-key, and would also ensure the success of the project.

***** Annual Maintenance *****

Please note that the Annual Maintenance is included for the first year and we recommend that it is renewed on an annual basis so that you continue to receive support and software updates.

***** Delivery Options *****

Please choose one option:

- Product CD sent to you by courier -- AU$ 50 + 3 days
- Software provided on FTP site to download -- FREE + 1 day
- Please provide your shipper information

***** Ordering Information *****

Please fax your purchase order or VISA/MASTERCARD information to:

Attn : Bob Erickson
fax +61 2 4960 1212

Matrikon Pty Ltd
(ABN: 21 002 695 534)
56 Kishorn Rd
Mt Pleasant WA 6153
Australia

You can also e-mail it directly to bob.erickson@matrikon.com

- Applicable Duties and Taxes are not included.
- Payments are due within 30 days of the date on the Matrikon invoice.

***** This Quotation is Valid for 30 Days *****

I am checking up on whether we still support a USB dongle for Tunneller. Please let me know if you have any other questions, thanks!

Best Regards,
Bob

Bob Erickson **BSc. EE, P.Eng (AB, Canada)**

Regional Manager
MatrikonOPC
dir: +61.8.9315.0115
fax: +61.2.4960.1212
mobile: 0408.428.498

MatrikonOPC Asia-Pacific
56 Kishorn Road
Mount Pleasant, WA 6153
Perth, Australia
www.matrikon.com

bob.erickson@matrikon.com

Appendix F

Visual Basic Code

F.1 Code Developed

```
'*************************************************************************
' Cycle time Analyser using OPC
' By C. Bollaart 'Robert Bosch (Australia)' 02/04/07
'*************************************************************************
' This project has been configured to reference the'OPC Automation 2.0' object.
' When modifying this project you must first add the 'OPC Automation 2.0' object
' to the reference object list. This can be done from the VB menu Project|References.
' Once the References Dialog is displayed, scroll down to the 'OPC Automation 2.0'
' object and select it, then click the OK button.  You'll know that the
' 'OPC Automation 2.0' object is being referenced when VB displays smart
' object properties for the OPC related objects as you use them.

'*************************************************************************
' 06/06/06 - Added to refernece the OPCDAAuto.dll
'*************************************************************************

Option Explicit
Option Base 1

' Server and group related data
' The OPCServer objects must be declared here due to the use of WithEvents
Dim WithEvents AnOPCServer As OPCServer
Dim WithEvents ConnectedOPCServer As OPCServer
Dim ConnectedServerGroup As OPCGroups
Dim WithEvents ConnectedGroup As OPCGroup

' OPC Item related data
Dim OPCItemCollection As OPCItems
Dim OneOPCItem As OPCItem
Dim ItemCount As Long
Dim OPCItemIDs(16) As String
Dim ItemServerHandles() As Long
Dim ItemServerErrors() As Long
Dim ClientHandles(16) As Long


Dim StartRecordingButton As Boolean
Dim Cycles As Integer
Dim Start As Boolean
Dim Delay As Boolean
Dim Trig As Integer
Dim Finish As Integer
```

```
Dim RecValues(0 To 10720) As Variant
Dim RecTimeStamp(0 To 10720) As Variant

Private Type SYSTEMTIMEREC
 wYear As Integer
 wMonth As Integer
 wDayOfWeek As Integer
 wDay As Integer
 wHour As Integer
 wMinute As Integer
 wSecond As Integer
 wMilliseconds As Integer
End Type

Private Declare Function SystemTimeToVariantTime Lib "oleaut32.dll" _
(ByVal vtime As Date, lpSystemTime As SYSTEMTIMEREC) As Long
Private Declare Function GetSystemTime Lib "kernel32" _
   (lpSystemTime As SYSTEMTIMEREC) As Long

Private Declare Function VariantTimeToSystemTime Lib "oleaut32.dll" _
(ByVal vtime As Double, lpSystemTime As SYSTEMTIMEREC) As Long

   'Private Declare Function VariantTimeToSystemTime Lib "kernel32" (ByVal vTime As Double,
ByRef lpSystemTime As SYSTEMTIMEITEM) As Long
Private Sub Command1_Click()
   Dim i As Integer
   Dim Dummy As String
        Open "c:\temp\ref.txt" For Output As #256
        For i = 0 To UBound(RecTimeStamp)
           Dummy = RecValues(i) & vbTab & RecTimeStamp(i)
           Print #256, Dummy
        Next i
        Close #256
End Sub

Private Sub Command2_Click()
Form1.Show
End Sub

Private Sub Command3_Click()
Form3.Show
End Sub

' General startup initialization
Private Sub Form_Load()
   AvailableOPCServerList.AddItem "Click on 'List OPC Servers' to start"
   Finish = 9
End Sub

' Make sure things get shut down properly upon closing application
Private Sub Form_Terminate()
   Call ExitExample_Click
End Sub


' This sub handles gathering a list of available OPC Servers and displays them
```

```
' The OPCServer Object provides a method called 'GetOPCServers' that will allow
' you to get a list of the OPC Servers that are installed on your machine.  The
' list is retured as a string array.
Private Sub ListOPCServers_Click()
   Dim AllOPCServers As Variant
   Dim i As Integer

   'Set error handling for OPC Function
   On Error GoTo ShowOPCGetServersError

   ' Create a temporary OPCServer object and use it to get the list of
   ' available OPC Servers
   Set AnOPCServer = New OPCServer
   ' Clear the list control used to display them
   AvailableOPCServerList.Clear
   'AllOPCServers = AnOPCServer.GetOPCServers("10.23.79.57")
   AllOPCServers = AnOPCServer.GetOPCServers(OPCNodeName.Text)
   ' Load the list returned into the List box for user selection
   For i = LBound(AllOPCServers) To UBound(AllOPCServers)
      AvailableOPCServerList.AddItem AllOPCServers(i)
   Next i

   GoTo SkipOPCGetServersError

ShowOPCGetServersError:
   Call DisplayOPC_COM_ErrorValue("Get OPC Server List", Err.Number)
SkipOPCGetServersError:
   ' Release the temporary OPCServer object now that we're done with it
   Set AnOPCServer = Nothing

End Sub

' This sub loads the OPC Server name when selected from the list
' and places it in the OPCServerName object
Private Sub AvailableOPCServerList_Click()
   ' When a user selects a server from the list box its name is placed
   ' in the OPCServerName
   OPCServerName = AvailableOPCServerList.List(AvailableOPCServerList.ListIndex)
End Sub


Private Sub Load_Click()
   Dim InString As String
   Dim Fname As String
   Dim txt As String
   Dim Meldg As Variant
   Dim Names() As String
   Dim i As Integer

   'MyArray As Variant
   'Open file with OPC tags
   Fname = Application.GetOpenFilename("Data file *.txt,*.txt,All Data *.*,*.*", , "Open")

   'If Fname = False Then Exit Sub
   'Read OPC tags from text file and place in array
   'Set mfile = mFileSysObj.GetFile("c:\data.txt")
```

```
    'Read OPC tags from text file and place in array
    'Set mTxtStream = mfile.OpenAsTextStream(ForReading)
    'lblFileName.Caption = mfile.Path

    'Dim InString As String, MyArray As Variant
    Open Fname For Input As #1
    InString = Input(LOF(1), #1)
    Close #1
    'now split the string
    Names = Split(InString, vbCrLf, -1)
          For i = 0 To UBound(Names)
             OPCItemName(i) = Names(i)
          Next i


     Exit Sub
ErrorHandler:
    txt = "Failure when opening file " & Fname & "."
    Meldg = MsgBox(txt, vbOKOnly)


          'Write Array to disk
          'Open "c:\temp\data.txt" For Output As #256
          'For i = 0 To UBound(RecTimeStamp)
          '    Dummy = RecValues(i) & vbTab & RecTimeStamp(i)
          '    Print #256, Dummy
          'Next i
          'Close #256
End Sub

' This sub handles connecting with the selected OPC Server
' The OPCServer Object provides a method called 'Connect' that allows you
' to 'connect' with an OPC server.  The 'Connect' method can take two arguments,
' a server name and a Node name.  The Node name is optional and does not have to
' be used to connect to a local server.  When the 'Connect' method is called you
' should see the OPC Server application start if it is not aleady running.
'
'Special Note: When connecting remotely to another PC running an OPC server make
'sure that you have properly configured DCOM on both PC's.
Private Sub OPCServerConnect_Click()
Dim ConnectedServerName As String
Dim ConnectedNodeName As Variant


' Test to see if the User has entered or selected an OPC server name yet if not post a message
If InStr(OPCServerName.Text, "Click") = 0 Then
    'Set error handling for OPC Function
    On Error GoTo ShowOPCConnectError
    '
    'Create a new OPC Server object
    Set ConnectedOPCServer = New OPCServer
    'Load the selected server name to start the interface
    ConnectedServerName = OPCServerName.Text
    'Load the node name of the connected server. The node name should be entered
    'without the use of forward slashes \\.
    ConnectedNodeName = OPCNodeName.Text
```

```
    'ConnectedNodeName = "10.23.79.57"
    'Attempt to connect with the server
    ConnectedOPCServer.Connect ConnectedServerName, ConnectedNodeName

    ' Throughout this example you will see a lot of code that simply enables
    ' and disables the various controls on the form.  The purpose of these
    ' actions is to demonstrate and insure the proper sequence of events when
    ' making an OPC connection.
    ' If we successfully connect to a server allow the user to disconnect
    DisconnectFromServer.Enabled = True
    ' Don't allow a reconnect until the user disconnects
    OPCServerConnect.Enabled = False
    AvailableOPCServerList.Enabled = False
    OPCServerName.Enabled = False

    ' Enable the group controls now that we have a server connection
    OPCGroupName.Enabled = True
    GroupUpdateRate.Enabled = True
    GroupDeadBand.Enabled = True
    GroupActiveState.Enabled = True
    AddOPCGroup.Enabled = True ' Remove group isn't enable until a group has been added

    GoTo SkipOPCConnectError

ShowOPCConnectError:
    DisconnectFromServer.Enabled = False
    Set ConnectedOPCServer = Nothing
    Call DisplayOPC_COM_ErrorValue("Connect", Err.Number)
SkipOPCConnectError:
Else
    ' A server name has not been selected yet post an error to the user
    Dim Response
    Response = MsgBox("You must first select an OPC Server, Click on the 'List OPC Servers' button and
select a server", vbOKOnly, "OPC Server Connect")
End If
End Sub

' This sub handles disconnecting from the OPC Server.  The OPCServer Object
' provides the method 'Disconnect'.  Calling this on an active OPCSerer
' object will release the OPC Server interface with your application.  When
' this occurs you should see the OPC server application shut down if it started
' automatically on the OPC connect. This step should not occur until the group
' and items have been removed
Private Sub DisconnectFromServer_Click()

' Test to see if the OPC Server connection is currently available
If Not ConnectedOPCServer Is Nothing Then
    'Set error handling for OPC Function
    On Error GoTo ShowOPCDisconnectError

    'Disconnect from the server, This should only occur after the items and group
    ' have been removed
    ConnectedOPCServer.Disconnect

    ' Release the old instance of the OPC Server object and allow the resources
    ' to be freed
```

```
    Set ConnectedOPCServer = Nothing

    ' Allow a reconnect once the disconnect completes
    OPCServerConnect.Enabled = True
    AvailableOPCServerList.Enabled = True
    OPCServerName.Enabled = True

    ' Don't alllow the Disconnect to be issued now that the connection is closed
    DisconnectFromServer.Enabled = False

    ' Disable the group controls now that we no longer have a server connection
    OPCGroupName.Enabled = False
    GroupUpdateRate.Enabled = False
    GroupDeadBand.Enabled = False
    GroupActiveState.Enabled = False
    AddOPCGroup.Enabled = False
End If

    GoTo SkipDisconnectError

ShowOPCDisconnectError:
    Call DisplayOPC_COM_ErrorValue("Disconnect", Err.Number)
SkipDisconnectError:
End Sub

' This sub handles adding the group to the OPC server and establishing the
' group interface.  When adding a group you can preset some of the group
' parameters using the properties '.DefaultGroupIsActive'
' and '.DefaultGroupDeadband'.  Set these before adding the group. Once the
' group has been successfully added you can change these same settings
' along with the group update rate on the fly using the properties on the
' resulting OPCGroup object.
Private Sub AddOPCGroup_Click()
    'Set error handling for OPC Function
    On Error GoTo ShowOPCGroupAddError

    'Prepare to add a group to the current OPC Server
    ' Get the group interface from the server object
    Set ConnectedServerGroup = ConnectedOPCServer.OPCGroups

    ' Set the desire active state for the group
    ConnectedServerGroup.DefaultGroupIsActive = GroupActiveState.Value

    'Set the desired percent deadband
    ConnectedServerGroup.DefaultGroupDeadband = Val(GroupDeadBand.Text)

    ' Add the group and set its update rate
    Set ConnectedGroup = ConnectedServerGroup.Add(OPCGroupName.Text)
    ' Set the update rate for the group
    ConnectedGroup.UpdateRate = Val(GroupUpdateRate.Text)

    ' ****************************************************************
    ' Mark this group to receive asynchronous updates via the DataChange event.
    ' This setting is IMPORTANT. Without setting '.IsSubcribed' to True your
    ' VB application will not receive DataChange notifications.  This will
    ' make it appear that you have not properly connected to the server.
```

```
    ConnectedGroup.IsSubscribed = True

    '*****************************************************************

    ' Now that a group has been added disable the Add group Button and enable
    ' the Remove group Button.  This demo application adds only a single group
    AddOPCGroup.Enabled = False
    OPCGroupName.Enabled = False
    RemoveOPCGroup.Enabled = True

    ' Enable the OPC item controls now that a group has been added
    OPCAddItems.Enabled = True
    Dim i As Integer
    For i = 0 To 15
      OPCItemName(i).Enabled = True
    Next i

    ' Disable the Disconnect Server button since we now have a group that must be removed first
    DisconnectFromServer.Enabled = False

    GoTo SkipAddGroupError

ShowOPCGroupAddError:
    Call DisplayOPC_COM_ErrorValue("Add Group", Err.Number)
SkipAddGroupError:

End Sub


Private Sub Option1_Click(Index As Integer)
Trig = Index

End Sub

Private Sub Option2_Click(Index As Integer)
Finish = Index
End Sub

' This sub handles removing a group from the OPC server, this must be done after
' items have been removed.  The 'Remove' method allows a group to be removed
' by name from the OPC Server.  If your application will maintains more than
' one group you will need to keep a list of the group names for use in the
' 'Remove' method.  In this demo there is only one group.  The name is maintained
' in the OPCGroupName TextBox but it can not be changed once the group is added.
Private Sub RemoveOPCGroup_Click()
' Test to see if the OPC Group object is currently available
If Not ConnectedServerGroup Is Nothing Then
    'Set error handling for OPC Function
    On Error GoTo ShowOPCGroupRemoveError

    ' Remove the group from the server
    ConnectedServerGroup.Remove (OPCGroupName.Text)
    ' Release the group interface and allow the server to cleanup the resources used
    Set ConnectedServerGroup = Nothing
    Set ConnectedGroup = Nothing
```

```
    ' Enable the Add group Button and disable the Remove group Button
    AddOPCGroup.Enabled = True
    OPCGroupName.Enabled = True
    RemoveOPCGroup.Enabled = False
    ' Disable the item controls now that a group has been removed.
    ' Items can't be added without a group so prevent the user from editing them.
    OPCAddItems.Enabled = False
    Dim i As Integer
    For i = 0 To 15
      OPCItemName(i).Enabled = False
    Next i

    ' Enable the Disconnect Server button since we have removed the group and can disconnect from the
server properly
    DisconnectFromServer.Enabled = True
End If

    GoTo SkipRemoveGroupError

ShowOPCGroupRemoveError:
    Call DisplayOPC_COM_ErrorValue("Remove Group", Err.Number)
SkipRemoveGroupError:
End Sub

' This sub allows the group's active state to be changed on the fly.  The
' OPCGroup object provides a number of properties that can be used to control
' a group's operation.  The '.IsActive' property allows you to turn all of the
' OPC items in the group On(active) and Off(inactive).
' Changing the actvie state of a group can be useful in controlling how your
' application makes use of an OPC Servers communication bandwidth.  If you don't
' need any of the data in a given group simply set it inactive, this will allow an
' OPC Server to gather only the data current required by your application.
Private Sub GroupActiveState_Click()
    'Set error handling for OPC Function
    On Error GoTo ShowOPCGroupActiveError

    ' If the group has been added and exist then change its active state
    If Not ConnectedGroup Is Nothing Then
       ConnectedGroup.IsActive = GroupActiveState.Value
    End If

    GoTo SkipGroupActiveError

ShowOPCGroupActiveError:
    Call DisplayOPC_COM_ErrorValue("Group Active State", Err.Number)
SkipGroupActiveError:

End Sub

' This sub allows the group's deadband to be changed on the fly.  Like the
' '.IsActive' property, the '.DeadBand' property can be changed at any time.
' The Deadband property allows you to control how much change must occur in
' an OPC item in this group before the value will be reported in the 'DataChange'
' event.  The value entered for '.DeadBand' is 0 to 100 as a percentage of full
' scale for each OPC item data type within this group.  If your OPC item is a
```

```
' Short(VT_I2) then your full scale is -32768 to 32767 or 65535.  If you
' enter a Deadband value of 1% then all OPC Items in this goup would need
' to change by a value of 655 before the change would be returned in the
' 'DataChange' event.  The '.DeadBand' property is a floating point number
' allowing very small ranges of change to be filtered.
Private Sub GroupDeadBand_Change()
    'Set error handling for OPC Function
    On Error GoTo ShowOPCGroupDeadBandError

    ' If the group has been added and exist then change its dead band
    If Not ConnectedGroup Is Nothing Then
        ConnectedGroup.DeadBand = Val(GroupDeadBand.Text)
    End If

    GoTo SkipGroupDeadBandError

ShowOPCGroupDeadBandError:
    Call DisplayOPC_COM_ErrorValue("Group Dead Band", Err.Number)
SkipGroupDeadBandError:
End Sub

' This sub allows the group's update rate to be changed on the fly.  The
' '.UpdateRate' property allows you to control how often data from this
' group will be returned to your application in the 'DataChange' event.
' The '.UpdateRate' property can be used to control and improve the overall
' performance of you application.  In this example you can see that the update
' rate is set for maximum update speed.  In a demo that's OK.  In your real
' world application, forcing the OPC Server to gather all of the OPC items in
' a group at their fastest rate may not be ideal.  In applications where you
' have data that needs to be acquired at different rates you can create
' multiple groups each with its own update rate.  Using multiple groups would
' allow you to gather time critical data in GroupA with an update rate
' of 200 millliseconds, and gather low priority data from GroupB with an
' update rate of 7000 milliseconds.  The lowest value for the '.UpdateRate'
' is 0 which tells the OPC Server go as fast as possible.  The maximium is
' 2147483647 milliseconds which is about 596 hours.
Private Sub GroupUpdateRate_Change()
    'Set error handling for OPC Function
    On Error GoTo ShowOPCGroupUdateRateError

    ' If the group has been added and exist then change its update rate
    If Not ConnectedGroup Is Nothing Then
        ConnectedGroup.UpdateRate = Val(GroupUpdateRate.Text)
    End If

    GoTo SkipGroupUdateRateError

ShowOPCGroupUdateRateError:
    Call DisplayOPC_COM_ErrorValue("Group Update Rate", Err.Number)
SkipGroupUdateRateError:
End Sub

' This sub handles adding an OPC item to a group.  The group must be established first before
' any items can be added.  Once you  have a group added to the OPC Server you
' need to add item to the group.  The OPCItems object provides the methods and
' properties need to add item to an estabished OPC group.
```

```
Private Sub OPCAddItems_Click()
   'Enable start recording button
   StartRecording.Enabled = True
   'Set error handling for OPC Function
   On Error GoTo ShowOPCItemAddError

   ' In this example we have at most 16 items that we will attempt to add
   ' to the group.
   ItemCount = 16
   '
   ' Load the request OPC Item names and build the ClientHandles list
   Dim i As Integer
   For i = 0 To 15
      ' Load the name of then item to be added to this group.  You can add
      ' as many items as you want to the group in a single call by building these
      ' arrays as needed.
      OPCItemIDs(i + 1) = OPCItemName(i).Text

      ' The client handles are given to the OPC Server for each item you intend
      ' to add to the group.  The OPC Server will uses these client handles
      ' by returning them to you in the 'DataChange' event.  You can use the
      ' client handles as a key to linking each valued returned from the Server
      ' back to some element in your application.  In this example we are simply
      ' placing the Index number of each control that will be used to display
      ' data for the item.  In your application the ClientHandle value you use
      ' can by whatever you need to best fit your program.  You will see how
      ' these client handles are used in the 'DataChange' event handler.
      ClientHandles(i + 1) = i

      ' Make the Items active start control Active, for the demo I want all itme to start active
      ' Your application may need to start the items as inactive.
      'OPCItemActiveState(i).Value = 1
   Next i

   ' Establish a connection to the OPC item interface of the connected group
   Set OPCItemCollection = ConnectedGroup.OPCItems

   ' Setting the '.DefaultIsActive' property forces all items we are about to
   ' add to the group to be added in an active state.  If you want to add them
   ' all as inactive simply set this property false, you can always make the
   ' items active later as needed using each item's own active state property.
   ' One key distinction to note, the active state of an item is independent
   ' from the group active state.  If a group is active but the item is
   ' inactive no data will be received for the item.  Also changing the
   ' state of the group will not change the state of an item.
   OPCItemCollection.DefaultIsActive = True

   ' Atempt to add the items,  some may fail so the ItemServerErrors will need
   ' to be check on completion of the call.  We are adding all item using the
   ' default data type of VT_EMPTY and letting the server pick the appropriate
   ' data type.  The ItemServerHandles is an array that the OPC Server will
   ' return to your application.  This array like your own ClientHandles array
   ' is used by the server to allow you to reference individual items in an OPC
   ' group.  When you need to perform an action on a single OPC item you will
   ' need to use the ItemServerHandles for that item.  With this said you need to
   ' maintain the ItemServerHandles array for use throughout your application.
```

```
    ' Use of the ItemServerHandles will be demonstrated in other subroutines in
    ' this example program.
    OPCItemCollection.AddItems ItemCount, OPCItemIDs, ClientHandles, ItemServerHandles,
ItemServerErrors

    ' This next step checks the error return on each item we attempted to
    ' register.  If an item is in error it's associated controls will be
    ' disabled.  If all items are in error then the Add Item button will
    ' remain active.
    Dim AnItemIsGood As Boolean
    AnItemIsGood = False
    For i = 0 To 15
        If ItemServerErrors(i + 1) = 0 Then ' If the item was added successfully then allow it to be used.
            'OPCItemValueToWrite(i).Enabled = True
            'OPCItemWriteButton(i).Enabled = True removed as write button deleted
            'OPCItemActiveState(i).Enabled = True
            'OPCItemSyncReadButton(i).Enabled = True
            AnItemIsGood = True
        Else
            ItemServerHandles(i + 1) = 0 ' If the handle was bad mark it as empty
            'OPCItemValueToWrite(i).Enabled = False
            'OPCItemWriteButton(i).Enabled = False
            'OPCItemActiveState(i).Enabled = False
            'OPCItemSyncReadButton(i).Enabled = False
        End If

    Next i

    ' Disable the Add OPC item button if any item in the list was good
    If AnItemIsGood Then
        OPCAddItems.Enabled = False
        For i = 0 To 15
            OPCItemName(i).Enabled = False ' Disable the Item Name cotnrols while now that they have
been added to the group.
        Next i
        RemoveOPCGroup.Enabled = False  ' If an item has been don't allow the group to be removed until
the item is removed
        OPCRemoveItems.Enabled = True
    Else
        ' The OPC Server did not accept any of the items we attempted to enter, let the user know to try
again.
        Dim Response
        Response = MsgBox("The OPC Server has not accepted any of the item you have enter, check your
item names and try again.", vbOKOnly, "OPC Add Item")
    End If

    GoTo SkipOPCItemAddError

ShowOPCItemAddError:
    Call DisplayOPC_COM_ErrorValue("OPC Add Items", Err.Number)
SkipOPCItemAddError:
End Sub

' This sub handles removing OPC items from a group.  Like the 'AddItems' method
' of the OPCItems object, the 'Remove' method allow us to remove item from
' an OPC group.  In this example we are removing all item from the group.
```

```
' In your application you may find it necessary to remove some items from
' a group while ading others.  Normally the best practice however it to add
' all the item you wish to use to the group and then control their active
' states individually.  You can control the active state of individual items
' in a group as shown in the 'OPCItemActiveState_Click' subroutine of this
' module.  With that said if you intend to remove the group you
' should first remove all its items.  The 'Remove' method uses the
' ItemServerHandles we received from the 'AddItems' method to properly remove
' only the items you wish.  This is an example of how ItemServerHandles are
' used by your application and the OPC Server.  As stated above, you can
' design your application to add and remove items as needed but that's not
' necessarily the most effiecent operation for the OPC Server.
Private Sub OPCRemoveItems_Click()
'Reset any recording before removing group
Start = False
StartRecordingButton = False
'Disable start recording button
StartRecording.Enabled = False
   If Not OPCItemCollection Is Nothing Then
      'Set error handling for OPC Function
      On Error GoTo ShowOPCRemoveItemError

      ItemCount = 1

      ' Provide an array to contain the ItemServerHandles of the item
      ' we intend to remove
      Dim RemoveItemServerHandles(16) As Long

      ' Array for potential error returns.  This example doesn't
      ' check them but yours should ultimately.
      Dim RemoveItemServerErrors() As Long

      ' Get the Servers handle for the desired items.  The server handles
      ' were returned in add item subroutine.  In this case we need to get
      ' only the handles for item that are valid.
      Dim i As Integer
      For i = 1 To 16
         ' In this example if the ItemServerHandle is non zero it is valid
         If ItemServerHandles(i) <> 0 Then
            RemoveItemServerHandles(ItemCount) = ItemServerHandles(i)
            ItemCount = ItemCount + 1
         End If
      Next i

      ' Item count is 1 greater than it needs to be at this point
      ItemCount = ItemCount - 1

      ' Invoke the Remove Item operation.  Remember this call will
      ' wait until completion
      OPCItemCollection.Remove ItemCount, RemoveItemServerHandles, RemoveItemServerErrors

      ' Clear the ItemServerHandles and turn off the controls for interacting
      ' with the OPC items on the form.
      For i = 0 To 15
         ItemServerHandles(i + 1) = 0 'Mark the handle as empty
         'OPCItemValueToWrite(i).Enabled = False
```

```
            'OPCItemWriteButton(i).Enabled = False removes as write buttoms deleted
            'OPCItemActiveState(i).Enabled = False
            'OPCItemSyncReadButton(i).Enabled = False
        Next i

        ' Enable the Add OPC item button and Remove Group button now that the
        ' items are released
        OPCAddItems.Enabled = True
        RemoveOPCGroup.Enabled = True
        OPCRemoveItems.Enabled = False

        ' Enable the OPC Item name controls to allow a new set of items
        ' to be entered
        For i = 0 To 15
            OPCItemName(i).Enabled = True
        Next i

        ' Release the resources by the item collection interface
        Set OPCItemCollection = Nothing

    End If

        GoTo SkipOPCRemoveItemError

ShowOPCRemoveItemError:
    Call DisplayOPC_COM_ErrorValue("OPC Remove Items", Err.Number)
SkipOPCRemoveItemError:

End Sub

' This sub handles writing a single value to the server using the
' 'SyncWrite' write method.  The 'SyncWrite' method provides a
' quick(programming wise) means to send a value to an OPC Server.  The item
' you intend to write must already be part of an OPC group you have added
' and you must have the ItemServerHandle for the item.  This is another example
' of how the ItemServerHandle is used and why it is important to properly
' store and track these handles.  The 'SyncWrite' method while quick and easy
' will wait for the OPC Server to complete the operation.  Once you invoke
' the 'SyncWrite' method it could take a moment for the OPC Server to return
' control to your application.  For this example that's OK.  If your application
' can't tolerate a pause you can use the 'AsyncWrite' and its associated
' 'AsyncWriteComplete' call back event instead.  In this sub we are only
' writing one value at a time.  The 'SyncWrite' mehtod can take a list of
' writes to be performed allow you to write entire recipes to the server
' in one shot.  If you are going to write more than one item, the
' ItemServerHandles for each item must be from the same OPC Group.
Private Sub OPCItemWriteButton_Click(Index As Integer)
    'Set error handling for OPC Function
    On Error GoTo ShowOPCSyncWriteError

    ' Write only 1 item
    ItemCount = 1
    ' Create some local scope variables to hold the value to be sent.
    ' These arrays could just as easily contain all of the item we have added.
    Dim SyncItemValues(1) As Variant
    Dim SyncItemServerHandles(1) As Long
```

```
    Dim SyncItemServerErrors() As Long

    ' Get the Servers handle for the desired item.  The server handles
    ' were returned in add item subroutine.
    SyncItemServerHandles(1) = ItemServerHandles(Index + 1)

    ' Load the value to be written
    'SyncItemValues(1) = Val(OPCItemValueToWrite(Index).Text)

    ' Invoke the SyncWrite operation.  Remember this call will wait until completion
    ConnectedGroup.SyncWrite ItemCount, SyncItemServerHandles, SyncItemValues,
SyncItemServerErrors

    GoTo SkipOPCSyncWriteError

ShowOPCSyncWriteError:
    Call DisplayOPC_COM_ErrorValue("OPC Sync Write", Err.Number)
SkipOPCSyncWriteError:

End Sub

' This sub handles performing a single synchronous read from a single item
' using the 'SyncRead' method.  The 'SyncRead' method like the 'SyncWrite',
' will wait for comletion from the server before returning to your application.
' There are two sources for data an OPC Server can return to your application.
' The first source is from 'Cache' the other is from 'Device'.  The 'SyncRead'
' method allows you to choose where you want to get the data.  If you choose
' 'Cache' the data has the potential to be old data which you can determine by
' looking at the time stamp on the data.  If you know that the data you are
' requesting is actively being scanned by the OPC Server you should be able to
' invoke the 'SyncRead' method using the mode selection of 'OPCCache'.  If your
' not sure if the data you desire is being scanned by the server or its out of
' date, you can use the mode selection of 'OPCDevice'.  The 'OPCDevice' mode
' commands the OPC Server to go and get this item directly from the device and
' 'DO IT NOW'.  This pretty much insures that you will receive the most recent
' value for the itme your are requesting.  The downside, when reading from the
' device directly the 'SyncRead' method will wait for the device to complete
' that read operation which could include mire time, modem time, or any other
' factor that is required to gather data from the actual device.  There are some
' benefits to using a 'SyncRead'  in the 'OPCDevice' mode.  If you want to
' completely control the data acquisition cycle from your application you can
' add your groups, add your items, make the items inactive, then use the 'SyncRead'
' mehtod to forcibly make the server perform read operations when you want.
' Using this scheme the server would only talk to the the device when you invoke
' either a 'SyncRead' or 'SyncWrite' method.
Private Sub SyncRecording()
    'Set error handling for OPC Function
    On Error GoTo ShowOPCSyncReadError
    Dim ItemCount As Long
    ' Provide storage the arrays returned by the 'SyncRead' method
    Dim ServerIndex As Long
    Dim Values() As Variant
    Dim ServerHandles(16) As Long
    Dim Errors() As Long
    Dim Quality As Variant
    Dim TimeStamp As Variant
```

```
    Dim i As Integer
    ItemCount = 16

    ' Get the Servers handle for the desired item.  The server handles were
    ' returned in add item subroutine.
    For ServerIndex = 1 To ItemCount
       ServerHandles(ServerIndex) = ItemServerHandles(ServerIndex)
    Next ServerIndex
    ' Invoke the SyncRead operation.  Remember this call will wait until
    ' completion. The source flag in this case, 'OPCDevice' , is set to
    ' read from device which may take some time.

     ConnectedGroup.SyncRead OPCDevice, ItemCount, ServerHandles, Values, Errors, Quality,
TimeStamp
    ' Convert variants to Arrays
    Dim Qualities() As Long
    ReDim Qualities(ItemCount)

    If VarType(Quality) = vbArray + vbInteger Then
    Dim Buffer() As Integer
    Buffer = Quality
    Dim ii As Integer
       For ii = 1 To ItemCount
       Qualities(ii) = Buffer(ii)
       Next ii
    End If

    Dim TimeStamps() As Double
    ReDim TimeStamps(ItemCount)

    If VarType(TimeStamp) = vbArray + vbDate Then
    Dim Buffer1() As Date
    Buffer1 = TimeStamp
    'Dim ii As Integer
       For ii = 1 To ItemCount
          TimeStamps(ii) = CDbl(Buffer1(ii))
       Next ii
    End If

    ' Save off the value returned after checking for error
    For ServerIndex = 1 To ItemCount
       If Errors(ServerIndex) = 0 Then
          'OPCItemValue(ServerIndex - 1).Text = Values(ServerIndex)
                If Values(ServerIndex) = True Then
                   OPCItemValue(ServerIndex - 1).Text = 1
                Else
                   OPCItemValue(ServerIndex - 1).Text = 0
                End If
          OPCItemQuality(ServerIndex - 1).Text = Qualities(ServerIndex)
          Dim Milliseconds As String
          Milliseconds = GetMilliseconds(TimeStamps(ServerIndex))
          OPCItemTimeStamp(ServerIndex - 1).Text = Milliseconds
          'OPCItemTimeStamp(ServerIndex - 1).Text = Format(TimeStamps(ServerIndex), "hh:mm:ss") &
"." & Format$(Milliseconds, "000")
          'OPCItemTimeStamp(ServerIndex - 1).Text = TimeStamps(ServerIndex)
```

```
        End If
    Next ServerIndex
    GoTo SkipOPCSyncReadError

ShowOPCSyncReadError:

    Call DisplayOPC_COM_ErrorValue("OPC Sync Read", Err.Number)
SkipOPCSyncReadError:
End Sub

' This Sub sets the active state of an individual item.  Like the other methods
' that perform operation on either a single item of list of items, the
' 'SetActive' method requires the ItemServerHandle of the item to be modified.
' Using the active state of an item allows you to control the amount of work
' the OPC Server is doing when communicating with a device.  You could add all
' the item you desire to read in an Active state but if some of those items are
' not currently in use you can improve the performance of the OPC Server by making
' those items inactive.
Private Sub OPCItemActiveState_Click(Index As Integer)
    'Set error handling for OPC Function
    On Error GoTo ShowOPCItemActiveStateError

    ' Change only 1 item
    ItemCount = 1
    ' Dim local arrays to pass desired item for state change
    Dim ActiveItemServerHandles(1) As Long
    Dim ActiveItemErrors() As Long
    Dim ActiveState As Boolean

    ' Get the desired state from the control.
    'ActiveState = OPCItemActiveState(Index).Value
    ' Get the Servers handle for the desired item.  The server handles
    ' were returned in add item subroutine.
    ActiveItemServerHandles(1) = ItemServerHandles(Index + 1)

    ' Invoke the SetActive operation on the OPC item collection interface
    OPCItemCollection.SetActive ItemCount, ActiveItemServerHandles, ActiveState, ActiveItemErrors

    ' Your application should check for errors here.

    GoTo SkipOPCItemActiveStateError

ShowOPCItemActiveStateError:
    Call DisplayOPC_COM_ErrorValue("OPC Item Active State", Err.Number)
SkipOPCItemActiveStateError:

End Sub
' This sub handles the 'DataChange' call back event which returns data that has
' been detected as changed within the OPC Server.  This call back should be
' used primarily to receive the data.  Do not make any other calls back into
' the OPC server from this call back.  The other item related functions covered
' in this example have shown how the ItemServerHandle is used to control and
' manipulate individual items in the OPC server.  The 'DataChange' event allows
' us to see how the 'ClientHandles we gave the OPC Server when adding items are
' used.  As you can see here the server returns the 'ClientHandles' as an array.
' The number of item returned in this event can change from trigger to trigger
```

```
' so don't count on always getting a 1 to 1 match with the number of items
' you have registered.  That where the 'ClientHandles' come into play.  Using
' the 'ClientHandles' returned here you can determine what data has changed and
' where in your application the data should go.  In this example the
' 'ClientHandles' were the Index number of each item we added to the group.
' Using this returned index number the 'DataChange' handler shown here knows
' what controls need to be updated with new data.  In your application you can
' make the client handles anything you like as long as they allow you to
' uniquely identify each item as it returned in this event.
Sub ConnectedGroup_DataChange(ByVal TransactionID As Long, ByVal NumItems As Long,
ClientHandles() As Long, ItemValues() As Variant, Qualities() As Long, TimeStamps() As Date)
    ' We don't have error handling here since this is an event called from the OPC interface
    Dim SYSTEMTIME As SYSTEMTIMEREC
    Dim stUTC As SYSTEMTIMEREC
    Dim i As Integer
    Dim Dummy1 As Variant
    Dim Dummy2 As Variant
    Dim Dummy3 As Variant
    Dim Hours As Integer
    Dim Minutes As Integer
    Dim Seconds As Integer
    Dim Milliseconds As Integer
    Dim Time As Variant
    For i = 1 To NumItems

    If SyncRead = 1 Then GoTo Finish1
        ' Use the 'Clienthandles' array returned by the server to pull out the
        ' index number of the control to update and load the value.
        If ItemValues(i) = True Then
        OPCItemValue(ClientHandles(i)).Text = 1
        Else
        OPCItemValue(ClientHandles(i)).Text = 0
        End If
        ' Find Milliseconds
        Milliseconds = GetMilliseconds(TimeStamps(i))
        OPCItemTimeStamp(ClientHandles(i)).Text = Format(TimeStamps(i), "hh:mm:ss") & "." &
Milliseconds
        ' Check the Qualties for each item retured here.  The actual contents of the
        ' quality field can contain bit field data which can provide specific
        ' error conditions.  Normally if everything is OK then the quality will
        ' contain the 0xC0
        If Qualities(i) And &HC0 Then
            OPCItemQuality(ClientHandles(i)).Text = "Good"
        Else
            OPCItemQuality(ClientHandles(i)).Text = Qualities(i)
        End If
    Next i
    Exit Sub
Crapped1:
    MsgBox "FileTimeToSystemTime Failed"
    Exit Sub
Finish1:
End Sub
' Handles displaying any OPC/COM/VB errors that are caught by the exception handler
Sub DisplayOPC_COM_ErrorValue(OPC_Function As String, ErrorCode As Long)
    Dim Response
```

```
    Dim ErrorDisplay As String
    ErrorDisplay = "The OPC function '" + OPC_Function + "' has returned an error of " + Str(ErrorCode)
+ " or Hex 0x" + Hex(ErrorCode)
    Response = MsgBox(ErrorDisplay, vbOKOnly, "OPC Function Error")
End Sub

' This sub handles exiting the program and properly disconnecting
' from the OPC server in an orderly fashion.  Like the force order
' of the controls on this form, the exit attempts to remove the Items
' from the group, then the group from the server and finally disconnect
' from the server.  This is also why each of the subroutines had the test
' to see if the Object to be removed was already set to 'Nothing'.
Private Sub ExitExample_Click()
    ' These calls will remove the OPC items, Group, and Disconnect in the proper order
    Call OPCRemoveItems_Click
    Call RemoveOPCGroup_Click
    Call DisconnectFromServer_Click
End
End Sub
' Module to memorise start recording activated
Private Sub StartRecording_Click()
    Start = False
    StartRecording.Enabled = False
    ' Set start recording memory
    StartRecordingButton = True
    ' Reset recording cycles counter
    Cycles = 0
    ProgressBar1.Value = 1
End Sub
' This sub sets the reading rate of Items
Private Sub Text4_Change()
Timer1.Interval = Text4.Text

End Sub
' This sub records all Item values and stores the results in file named
' C:\temp\data.text.
Private Sub Timer1_Timer()
' Declarations
Dim Dummy As String
Dim Y As Integer
Dim z As Integer
Dim X As Integer
Dim SYSTEMTIME As SYSTEMTIMEREC
    ' Wait for start recurding button is pressed
    If StartRecordingButton Then
        If SyncRead = 0 Then GoTo Jump1
        Call SyncRecording  ' read Items from OPC server
Jump1:
        ' Wait for Start signal from trigger timer
        If Start Then
            ' Variables
            Dim i As Integer
            i = Cycles * 16
            Y = 0
            X = i
            z = i + 15
```

```
            ' Copy 16 bits to record array
            For i = X To z
               RecValues(i) = CStr(OPCItemValue(Y).Text)
               RecTimeStamp(i) = OPCItemTimeStamp(Y).Text
               Y = Y + 1
            Next i
            ' Increment cycle counter
            Cycles = Cycles + 1
            ProgressBar1.Value = Cycles
         End If
      End If
      ' End recording when 670 samples have been taken
      If Cycles = 670 Then
         StartRecordingButton = False
         Start = False
         Cycles = 0
         Start = False
' Analyse results to determine time between trigger and finish
         ' Call sub to analyse cycle time
         AnalyseCycleTime

         For i = 0 To 15
            ' Call sub to analyse sub cycle for all 16 bits
            AnalyseSubCycle (i)
         Next i

         'Write Array to disk
         Open "c:\temp\data.txt" For Output As #256
            For i = 0 To UBound(RecTimeStamp)
               Dummy = RecValues(i) & vbTab & RecTimeStamp(i)
               Print #256, Dummy
            Next i
         Close #256
         ' Enable startRecording button
         StartRecording.Enabled = True
      End If
   Exit Sub
   Crapped1:
      MsgBox "FileTimeToSystemTime Failed"
      Exit Sub
   End Sub
   ' Timer module todetect trigger signal after start recording is activated
   Private Sub Timer2_Timer()
      ' Poll trigger item for true state
      If OPCItemValue(Trig).Text = "1" Then
         Start = True ' Set start bit
      End If
   End Sub
   ' This sub extracts the millisecond value from a UTC based timestamp
   Function GetMilliseconds(ByVal varDateTime As Variant) As String
   ' Declare variables
   Dim decConversionFactor As Variant
   Dim Dummy1 As Double
   Dim Dummy2 As Double
   Dim Dummy3 As Double
   Dim TotalmSec As Long
```

```
Dim Hours As Integer
Dim Minutes As Long
Dim Seconds As Long
Dim Milliseconds As Long
Dim decTime As Variant
' main
   Dummy1 = CDbl(varDateTime)
   Dummy2 = Fix(varDateTime)
   Dummy3 = varDateTime - Dummy2
   Dummy3 = Dummy3 * 100000000
   Dummy3 = Int(Dummy3)
   TotalmSec = Dummy3
   Hours = Fix(Int(Dummy3 / 3600000))
   Dummy3 = TotalmSec
   Minutes = Fix(((TotalmSec - (Hours * 3600000)) / 60000))
   Dummy3 = TotalmSec
   Seconds = Fix(((TotalmSec - (Hours * 3600000) - (Minutes * 60000)) / 1000))
   Milliseconds = Fix((TotalmSec - (Hours * 3600000) - (Minutes * 60000) - (Seconds * 1000)))
   GetMilliseconds = Format$(Hours, "00") & ":" & Format$(Minutes, "00") & ":" & Format$(Seconds,
"00") & "." & Format$(Milliseconds, "000")

End Function
' This sub handles the display of start time, end time and overall cycle time
' It uses the data already recorded and searches the RecValues array for the first low
' to high transition on the trigger input and then searches for a low to high transition
' on the end signal. The overall cycle time is then the difference between these 2 results.
Function AnalyseCycleTime()
Dim StartTime As Variant
Dim EndTime As Variant
Dim i As Integer
Dim MinuteStart As Variant
Dim SecondStart As Variant
Dim MillisecondStart As Variant
Dim MinuteEnd As Variant
Dim SecondEnd As Variant
Dim MillisecondEnd As Variant
Dim Minute As Variant
Dim Second As Variant
Dim Millisecond As Variant
Dim TotalMillisecondStart As Variant
Dim TotalSecondsStart As Variant
Dim TotalMillisecondEnd As Variant
Dim TotalSecondsEnd As Variant

' Search record array for time of trigger start, save start time in start field
Dim TestValue As String
TestValue = 1

      For i = Trig To UBound(RecValues)
      If RecValues(i) = TestValue Then
         Text1.Text = RecTimeStamp(i)
   Exit For     'stop searching
   End If
   i = i + 15
Next i
```

```
' Search record array for time of end signal, save end time in end field
   For i = Finish To UBound(RecValues)
      If RecValues(i) = TestValue Then
         Text2.Text = RecTimeStamp(i)
         Exit For     'stop searching
      Else
         ' If no value found display zero time
         Text2.Text = "00:00:00.000"

   End If
   i = i + 15
Next i

' If no end time was found ie end = zero time force start time to zero
If Text2.Text = "00:00:00.000" Then
  Text1.Text = "00:00:00.000"
End If

' Calculate seconds for start
MinuteStart = CDec(Left(Right(Text1.Text, 9), 2))
SecondStart = CDec(Left(Right(Text1.Text, 6), 2))
MillisecondStart = CDec(Right(Text1.Text, 3))
TotalMillisecondStart = MinuteStart * 60 * 1000 + SecondStart * 1000 + MillisecondStart
TotalSecondsStart = TotalMillisecondStart / 1000

' Calculate seconds for end
MinuteEnd = CDec(Left(Right(Text2.Text, 9), 2))
SecondEnd = CDec(Left(Right(Text2.Text, 6), 2))
MillisecondEnd = CDec(Right(Text2.Text, 3))
TotalMillisecondEnd = MinuteEnd * 60 * 1000 + SecondEnd * 1000 + MillisecondEnd
TotalSecondsEnd = TotalMillisecondEnd / 1000

' Record overall cycle time in seconds field
Second = TotalSecondsEnd - TotalSecondsStart
Text3.Text = Second

End Function
' This sub handles the analysis of sub cycles.
' It uses the data already recorded to calculate the period of time each signal
' remained in the high state the values calculated are displayed on the cycle times
' result form.
Function AnalyseSubCycle(X As Integer)
Dim StartTime As Variant
Dim EndTime As Variant
Dim i As Integer
Dim Y As Integer
Dim MinuteStart As Variant
Dim SecondStart As Variant
Dim MillisecondStart As Variant
Dim MinuteEnd As Variant
Dim SecondEnd As Variant
Dim MillisecondEnd As Variant
Dim Minute As Variant
Dim Second As Variant
Dim Millisecond As Variant
Dim TotalMillisecondStart As Variant
```

```
Dim TotalSecondsStart As Variant
Dim TotalMillisecondEnd As Variant
Dim TotalSecondsEnd As Variant

Dim Dummy1 As String
Dim Dummy2 As String
Dim Dummy3 As String

' Search record array for time of first high signal
Dim TestValue As String
i = X
For Y = 1 To 12
   TestValue = 1

         For i = i To UBound(RecValues)
            If RecValues(i) = TestValue Then
               Dummy1 = RecTimeStamp(i)
               Exit For     'stop searching
            Else
               Dummy1 = "00:00:00.000"
            End If
         i = i + 15
         Next i
   If i >= UBound(RecValues) Then ' If end of array is reached end
      GoTo Finish
   End If
' Search record array for time of, high to low transition
   i = i + 16
   TestValue = 0
         For i = i To UBound(RecValues)
            If RecValues(i) = TestValue Then
               Dummy2 = RecTimeStamp(i)
               Exit For     'stop searching
            Else
               Dummy2 = "00:00:00.000"
            End If
         i = i + 15
         Next i

   If Dummy2 = "00:00:00.000" Then
      Dummy1 = "00:00:00.000"
   End If
' Calculate seconds for start
   MinuteStart = CDec(Left(Right(Dummy1, 9), 2))
   SecondStart = CDec(Left(Right(Dummy1, 6), 2))
   MillisecondStart = CDec(Right(Dummy1, 3))
   TotalMillisecondStart = MinuteStart * 60 * 1000 + SecondStart * 1000 + MillisecondStart
   TotalSecondsStart = TotalMillisecondStart / 1000

' Calculate seconds for end
   MinuteEnd = CDec(Left(Right(Dummy2, 9), 2))
   SecondEnd = CDec(Left(Right(Dummy2, 6), 2))
   MillisecondEnd = CDec(Right(Dummy2, 3))
   TotalMillisecondEnd = MinuteEnd * 60 * 1000 + SecondEnd * 1000 + MillisecondEnd
   TotalSecondsEnd = TotalMillisecondEnd / 1000
```

```
' Calculate difference between start and end transitions
   Second = TotalSecondsEnd - TotalSecondsStart

' Record times on form and compare with best case and tolerance
   Select Case Y
   Case 1
      Form1.Sub1(X) = Second
      If CDec(Form1.Sub1(X).Text) > (CDec(Form3.Sub1(X).Text) - CDec(Form2.Sub1(X).Text)) And _
         CDec(Form1.Sub1(X).Text) < (CDec(Form3.Sub1(X).Text) + CDec(Form2.Sub1(X).Text)) Then
         Form1.Sub1(X).BackColor = &HC000&
         Sub1(X).BackColor = &HC000&
      Else
         Form1.Sub1(X).BackColor = &HFF&
         Sub1(X).BackColor = &HFF&
      End If
      Sub1(X) = Second
   Case 2
      Form1.Sub2(X) = Second
      If CDec(Form1.Sub2(X).Text) > CDec(Form3.Sub2(X).Text) - CDec(Form2.Sub2(X).Text) And _
         CDec(Form1.Sub2(X).Text) < CDec(Form3.Sub2(X).Text) + CDec(Form2.Sub2(X).Text) Then
         Form1.Sub2(X).BackColor = &HC000&
         Sub2(X).BackColor = &HC000&
      Else
         Form1.Sub2(X).BackColor = &HFF&
         Sub2(X).BackColor = &HFF&
      End If
      Sub2(X) = Second
   Case 3
      Form1.Sub3(X) = Second
      If CDec(Form1.Sub3(X).Text) > CDec(Form3.Sub3(X).Text) - CDec(Form2.Sub3(X).Text) And _
         CDec(Form1.Sub3(X).Text) < CDec(Form3.Sub3(X).Text) + CDec(Form2.Sub3(X).Text) Then
         Form1.Sub3(X).BackColor = &HC000&
         Sub3(X).BackColor = &HC000&
      Else
         Form1.Sub3(X).BackColor = &HFF&
         Sub3(X).BackColor = &HFF&
      End If
      Sub3(X) = Second
   Case 4
      Form1.Sub4(X) = Second
      If CDec(Form1.Sub4(X).Text) > CDec(Form3.Sub4(X).Text) - CDec(Form2.Sub4(X).Text) And _
         CDec(Form1.Sub4(X).Text) < CDec(Form3.Sub4(X).Text) + CDec(Form2.Sub4(X).Text) Then
         Form1.Sub4(X).BackColor = &HC000&
         Sub4(X).BackColor = &HC000&
      Else
         Form1.Sub4(X).BackColor = &HFF&
         Sub4(X).BackColor = &HFF&
      End If
      Sub4(X) = Second
   Case 5
      Form1.Sub5(X) = Second
      If CDec(Form1.Sub5(X).Text) > CDec(Form3.Sub5(X).Text) - CDec(Form2.Sub5(X).Text) And _
         CDec(Form1.Sub5(X).Text) < CDec(Form3.Sub5(X).Text) + CDec(Form2.Sub5(X).Text) Then
         Form1.Sub5(X).BackColor = &HC000&
         Sub5(X).BackColor = &HC000&
      Else
```

```
            Form1.Sub5(X).BackColor = &HFF&
            Sub5(X).BackColor = &HFF&
        End If
        Sub5(X) = Second
    Case 6
        Form1.Sub6(X) = Second
        If CDec(Form1.Sub6(X).Text) > CDec(Form3.Sub6(X).Text) - CDec(Form2.Sub6(X).Text) And _
            CDec(Form1.Sub6(X).Text) < CDec(Form3.Sub6(X).Text) + CDec(Form2.Sub6(X).Text) Then
            Form1.Sub6(X).BackColor = &HC000&
            Sub6(X).BackColor = &HC000&
        Else
            Form1.Sub6(X).BackColor = &HFF&
            Sub6(X).BackColor = &HFF&
        End If
        Sub6(X) = Second
    Case 7
        Form1.Sub7(X) = Second
        If CDec(Form1.Sub7(X).Text) > CDec(Form3.Sub7(X).Text) - CDec(Form2.Sub7(X).Text) And _
            CDec(Form1.Sub7(X).Text) < CDec(Form3.Sub7(X).Text) + CDec(Form2.Sub7(X).Text) Then
            Form1.Sub7(X).BackColor = &HC000&
            Sub7(X).BackColor = &HC000&
        Else
            Form1.Sub7(X).BackColor = &HFF&
            Sub7(X).BackColor = &HFF&
        End If
        Sub7(X) = Second
    Case 8
        Form1.Sub8(X) = Second
        If CDec(Form1.Sub8(X).Text) > CDec(Form3.Sub8(X).Text) - CDec(Form2.Sub8(X).Text) And _
            CDec(Form1.Sub8(X).Text) < CDec(Form3.Sub8(X).Text) + CDec(Form2.Sub8(X).Text) Then
            Form1.Sub8(X).BackColor = &HC000&
            'Sub8(X).BackColor = &HC000&
        Else
            Form1.Sub8(X).BackColor = &HFF&
            'Sub8(X).BackColor = &HFF&
        End If
    Case 9
        Form1.Sub9(X) = Second
        If CDec(Form1.Sub9(X).Text) > CDec(Form3.Sub9(X).Text) - CDec(Form2.Sub9(X).Text) And _
            CDec(Form1.Sub9(X).Text) < CDec(Form3.Sub9(X).Text) + CDec(Form2.Sub9(X).Text) Then
            Form1.Sub9(X).BackColor = &HC000&
        Else
            Form1.Sub9(X).BackColor = &HFF&
        End If
        'Form1.Sub9(X) = Second
    Case 10
        Form1.Sub10(X) = Second
        If CDec(Form1.Sub10(X).Text) > CDec(Form3.Sub10(X).Text) - CDec(Form2.Sub10(X).Text) And
_
            CDec(Form1.Sub10(X).Text) < CDec(Form3.Sub10(X).Text) + CDec(Form2.Sub10(X).Text)
Then
            Form1.Sub10(X).BackColor = &HC000&
        Else
            Form1.Sub10(X).BackColor = &HFF&
        End If
        'Form1.Sub10(X) = Second
```

```
   Case 11
      Form1.Sub11(X) = Second
      If CDec(Form1.Sub11(X).Text) > (Form3.Sub11(X).Text) - CDec(Form2.Sub11(X).Text) And _
         CDec(Form1.Sub11(X).Text) < (Form3.Sub11(X).Text) + CDec(Form2.Sub11(X).Text) Then
         Form1.Sub11(X).BackColor = &HC000&
      Else
         Form1.Sub11(X).BackColor = &HFF&
      End If
      'Form1.Sub11(X) = Second
   Case 12
      Form1.Sub12(X) = Second
      If CDec(Form1.Sub12(X).Text) > CDec(Form3.Sub12(X).Text) - CDec(Form2.Sub12(X).Text) And
_
         CDec(Form1.Sub12(X).Text) < CDec(Form3.Sub12(X).Text) + CDec(Form2.Sub12(X).Text)
Then
         Form1.Sub12(X).BackColor = &HC000&
      Else
         Form1.Sub12(X).BackColor = &HFF&
      End If
      'Form1.Sub12(X) = Second
   End Select
Next Y

Finish:
End Function

' Show Tolerance form t
Private Sub Tol_Click()
Form2.Show
End Sub
```