ISSN 0280-5316
ISRN LUTFD2/TFRT--5732--SE

Control Loop Performance Monitor

Fredrik Holstein

Department of Automatic Control
Lund Institute of Technology
December 2004

Department of Automatic Control
Lund Institute of Technology
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
December 2004

Document Number
ISRNLUTFD2/TFRT--5732--SE

Author(s)
Fredrik Holstein

Supervisor
Alexander Horch at ABB in Västerås
Tore Hägglund LTH in Lund

Sponsoring organization

Title and subtitle
Control Loop Performance Monitor (Regulatorövervakningsmonitor)

Abstract
In this thesis a control loop asset monitor is developed for ABB's automation system 800xA. The asset monitor developed here is based on ABB's Loop Performance Manager, which also is described here as well as the general properties and functions of an asset monitor. The asset monitor is a prototype for the 800xA system and a detailed description of how to implement asset monitors into the system is given. The control loop asset monitor was tested on stored data from real processes as well as on a lab scale process. This asset monitor will be used as a base for further control loop asset monitors and the report will help further asset monitor development in the 800xA system.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
75

Recipient's notes

Security classification

Contents

1 Introduction **6**

2 Control Performance Monitoring **8**
2.1 General 8
2.2 Optimize^{IT} LPM 8

3 Asset Optimization **16**
3.1 General 16
3.2 800xA Asset Optimization 17
3.3 Asset monitor 18

4 Control loop asset monitor **19**
4.1 Asset monitor SDK 19
4.2 Implementing the asset monitor 20
4.3 Simple asset monitor example 20
4.4 The LPM asset monitor 40
4.5 Testing the LPM Asset Monitor 43

5 Applications **47**
5.1 Simulink environment 47
5.2 Lab process with industrial automation 48

6 Final conclusions **53**

A Indices for LPM **55**

B Hypotheses for LPM **57**

C Visual Basic code Simple example **58**

D Matlab functions in the LPM Asset Monitor **61**

E Visual Basic code LPM Asset Monitor **63**

List of Figures

2.1 Thresholds for diagnosis. 11
2.2 Oscillating control loop 13
2.3 Well performing control loop 14
2.4 Control loop in manual mode 15

3.1 Maintenance labor usage. 16
3.2 800xA, platform for the complete integration of all data. 17
3.3 800xA Plant Explorer. 18

4.1 Implementation steps. 19
4.2 Step one: Matlab code. 21
4.3 The COM Builder GUI. 22
4.4 New Project. 23
4.5 Step two: Asset monitor structure in Excel. 24
4.6 Asset Monitor definition. 24
4.7 Conditions. 25
4.8 Asset Parameters. 26
4.9 Input Records. 26
4.10 Output Records. 27
4.11 Step three: Main logic code in Visual Basic. 27
4.12 Visual Basic properties. 28
4.13 Visual Basic references. 29
4.14 View of Visual Basic window. 30
4.15 Step four: Asset monitor creation 800xA. 33
4.16 Adding a new object. 34
4.17 Adding a Alarm and Event List aspect. 35
4.18 Adding a general properties aspect. 36
4.19 Config View of asset monitor. 37
4.20 Configure path to input records. 37
4.21 Set the input records. 38
4.22 Starting the server. 38
4.23 Testing the asset monitor. 39
4.24 Output records in 800xA. 39
4.25 Conditions for LPM asset monitor. 41

4.26 Asset parameters for LPM asset monitor. 41
4.27 Input records for LPM asset monitor. 42
4.28 Output records for LPM asset monitor. 42
4.29 Oscillating control loop data 44
4.30 Oscillating control loop alarms. 44
4.31 Ok control loop data 45
4.32 Ok control loop alarms. 45
4.33 Saturated control loop data 46
4.34 Saturated control loop alarms. 46

5.1 Data for stiction model 47
5.2 Stiction alarms. 48
5.3 Lab process as seen in the 800xA plant explorer. 49
5.4 Lab process. 49
5.5 Lab process data 50
5.6 Lab process alarms. 50
5.7 Lab process data 51
5.8 Lab process alarms. 52

A.1 Indices in the LPM. 56

B.1 Hypotheses in the LPM. 57

Chapter 1

Introduction

This thesis reports work done at ABB Corporate Research in Ladenburg, Germany. ABB's two main businesses are power technologies and automation technologies. ABB Power Technologies provide products, systems and services for power transmission, distribution and automation. ABB Automation Technologies deliver solutions for control, motion, protection and plant integration across the full range of process and utility industries.

The newest automation system from ABB is called **ABB's Industrial^{IT} Extended Automation System 800xA** and includes a component called **Asset Optimization**. This component contains tools for tuning, improving or optimizing productive systems or end products, which all leads to reduced maintenance.

Maintenance is a major expense for all companies today, which is the reason why considerable savings are possible in this area. Maintenance is carried out when something is broken or when something is found to be working badly at routine checks. This could mean high costs if processes have to be shut down to be able to repair what is broken. If the problems could be identified at an early stage this might be avoided and both costs and work can be reduced.

One of the core functionalities in ABB's 800xA Asset Optimization is asset monitoring. The main purposes of an asset monitor are to identify equipment failure, performance degradation and efficiency degradation that may impact production output or quality. The purpose of the work done in this thesis was to implement an asset monitor, which can monitor various control loops. The asset monitor developed is a prototype for the 800xA system and should be simple, but still work sufficiently well. Due to the prototype character of the implementation, examples of how to implement asset monitors into the 800xA system will also be given, in order to facilitate later implementations and improvements.

The asset monitor developed here evaluates certain control loop performance indices that are calculated from data, collected online from the monitored control loop. These indices give an indication of how well the control loop is currently performing.

In order to implement the asset monitor into the 800xA system, the different performance indices first had to be determined and then implemented in **Matlab**. This code was then used to implement the asset monitor in **Visual Basic**. The visual basic code constitutes the main logic of the asset monitor in 800xA. This together with the asset monitor structure, that is constructed in the **Asset Monitor Software Development Kit**, forms the asset monitor.

To be able to understand how the asset monitor should function, control performance monitoring, asset optimization and the 800xA system are explained in Chapters 2 and 3. Then the steps to implement an asset monitor into the 800xA system are described through an example in Chapter 4. In this Chapter the main asset monitor that is developed for this thesis, called the LPM Asset Monitor, is also described. The asset monitor was tested on stored data values and compared to earlier tests to this data. Finally, described in Chapter 5, the asset monitor was tested online, both on simulated data and on data from a lab scale process.

Chapter 2

Control Performance Monitoring

2.1 General

Most controllers in industries today are P-, PI- and PID controllers. These are often only tuned once, during commissioning, or sometimes even not at all. To be able to spot a badly working controller the plant engineer has to be experienced and has to monitor the control loop closely, which can be hard if many control loops have to be monitored, which usually is the case. If the detection of badly performing control loops could be made automatic a lot of resources could be saved and the quality of the control loop would increase.

The problems with automatic control loop monitors are to obtain information about the control loop by a passive monitoring tool and how to quantify what kind of behaviour that is deemed acceptable [2]. The industry also demands the following of an control loop performance monitor:

- Non invasiveness, it should be a passive observer.
- No new sensors, should only use what is already available.
- Minimal process knowledge, ideally assume no knowledge.
- Simple algorithms, should monitor many loops at the same time.

There are some control performance monitoring tools on the market, achieving these goals. One of them being the Loop Performance Manager tool from ABB.

2.2 Optimize^{IT} LPM

ABB's **Optimize^{IT} Loop Performance Manager (LPM)** is a product that helps monitor and tune process control loops [1]. The LPM contains an au-

diting module that performs control loop monitoring. The performance of the monitored control loops is evaluated through certain indices in the LPM. Each index contains a value that describes how a certain part of the control loop is behaving. Some of the indices are direct measures of how the control loop is performing, some carry more indirect information about the control loop. The LPM evaluates 38 indices. Some of these indexes are described in this chapter. From the indices a diagnosis is made containing 11 hypotheses describing the performance of the control loop, for example the hypothesis *control loop oscillating* can be given the evaluation value true, false or undecided.

The LPM is given a batch of data from the monitored control loop and then evaluates its performance and creates a report of the evaluation. The report can then be viewed, so that if the control loop is not working satisfactory it can be corrected. The different hypotheses are described at the end of this chapter.

2.2.1 Indices

The LPM calculates values for 38 indices that all in some way or other describe how the control loop is performing. Some of the more important indices are described here. For a short description of all the indices see Appendix A.

Standard deviation

An easy value to calculate for the collected data batch is the standard deviation of the process error. The absolute value of the standard deviation itself does not indicate bad behaviour, but the trend of it between batches would. If the standard deviation were to increase in time, it would mean that the controller is working worse and worse and should be scheduled for an inspection.

Oscillation index/amplitude

A major problem with control loops is oscillation. This will give bad performance and excessive controller movement, which increases the wear on the machinery. The integrated error between set point crossings are calculated and compared to each other. If there are several similar area sizes and times between set point crossings there is oscillation present. This is a very important index, since there is much to be gained if one can remove oscillations from a control loop. The amplitude is estimated as well together with the oscillation frequency.

Automatic/saturated mode

To be able to use the calculated indices at all, the mode of the control loop has to be evaluated first. If the control loop has been in manual mode or if it has been saturated for a longer time this may affect the values of the other indices. If the mode has been in manual for too long time the value of the other indices are not considered valid, since the controller has to be active to be able to evaluate its performance. A control loop that is saturated too much,

probably has to be readjusted so that it moves within its ranges. To see if the controller is in manual mode or not, adjacent control loop values are evaluated, if they do not change over several data points the controller is considered to be in manual mode during this time. If these data points are above or below certain limits, the loop is considered to be in saturated mode. The automatic index gives a percentage of how much of the data points in the batch that are in automatic mode. This has to be sufficiently large so that the other indices give an accurate picture of how the controller is performing. The saturated index gives, in the same way, a percentage of how many data points are saturated. If this value is too large the controller values should be reconfigured.

Valve travel

This is an index that gives a value of how much the control signal moves during the data batch. If the actuator varies a lot it means that a lot of extra energy is spent to keep the process variable at the desired value. This could be avoided by readjusting the controller parameters. The index gives a count of how many times the control signal has moved a percentage of its range. The bigger the value the more the actuator is working.

Setpoint crossing

If a controller is working well the process variable should be close to the set point. This means that the process error should be small and that the process variable will cross over the set point often. If there are few set point crossings, reasons could be sluggish control, oscillations or an offset (e.g. pure P-control).

Shut off PV

An important monitor for flow loops is the shut off process variable index. This index is the value of the process variable at times when the control signal is considered to be zero, meaning that the controller is shut off, so there should not be any flow through the loop. If there is a leakage in the loop the process variable will have a value greater than zero. The Shut off PV index gives a mean value of the process variable for all the times the control signal is zero, or very small. Small leakage may be expected, but if the value is over a legal limit, an alarm should be activated.

Harris index

To get an index for how good the controller of the process is performing the Harris index is calculated [3]. This index compares how big the standard deviation of the control error is to a minimum variance controller. The minimum achievable variance is estimated from the data that is collected. To be able to estimate a minimum variance controller for the process the dead time has to be known. The dead time is generalized to different kinds of control loops, e.g. liquid flow loops.

2.2.2 Diagnosis

After the indices are calculated a diagnosis is to be made whether the control loop is working satisfactory or not (and if not: why). To be able to make the diagnosis, sensible limits for the different indices have to be set. These limits are generalised for every group of controllers, every loop category. Some of the limits can be seen in Figure 2.1.

Threshold values for CLPA diagnoses

loop type / index	T1 Normalised error standard deviation limit	T2 Oscillation threshold control error	T3 Oscillation amplitude threshold	T4 Loop in auto	T5 Loop saturated	T6 Valve travel per hour	T7 Percentage of setpoint crossings	T8 Performance Index	T9 Shut-off PV value
Flow_liquid	0.5	0.3	0.1	60	10	500	10	0.6	2
Flow_other	0.5	0.3	0.1	60	10	500	10	0.6	2
Temperature	0.5	0.3	0.1	60	10	200	5	0.6	NaN
Composition	1	0.3	0.1	60	10	200	10	0.6	2
Pressure	1	0.3	0.1	60	10	200	10	0.6	NaN
Level_tight	1	0.3	0.1	60	10	500	10	0.6	NaN
Level_average	20	0.3	0.1	60	10	500	3	0.4	NaN
otherSRP	1	0.3	0.1	60	10	500	10	0.6	NaN
otherINT	1	0.3	0.1	60	10	200	10	0.6	NaN

Figure 2.1: Thresholds for diagnosis.

The diagnosis is made by evaluating the following 11 hypotheses, they can be true (1), false (0) or undecided (-1) and are based on one or several of the indices compared to their limits. The *Valve Problem* hypothesis is based on several indices. If one of the indices does not indicate valve problems while the others do, it is not certain that there are valve problems. Therefore this hypothesis has two different values of true, depending on if all indices indicate valve problems or not.

H1: Sluggish tuning
H2: Oscillating loop
H3: Possible valve problems
H4: Valve leakage or instrument zero error
H5: Oscillating setpoint
H6: Excessive valve movement
H7: Excessive measurement noise
H8: Acceptable tracking behaviour
H9: Basic statistics o.k.
H10: Controller in saturation
H11: Acceptable overall performance

All of the hypothesis can not always be calculated. E.g. the valve leakage hypothesis can only be calculated for control loops that monitor a flow. To be

sure that no leakage is present, the control signal and the process variable both have to be zero for sufficiently many of the data points. If the control signal is not turned off a certain time of the monitored interval, the hypothesis can not be evaluated.

The last hypothesis is based on eight preconditions. All off the preconditions have to be true for the overall performance to be good. The preconditions are as follows.

P1: Harris index good
P2: Set point crossing large enough
P3: Loop not oscillating
P4: Loop not saturated
P5: Loop not in manual mode
P6: Tracking acceptable
P7: No valve leakage
P8: No sluggish tuning

For an overlook of the hypotheses see Appendix B.

2.2.3 Testing the LPM

This section presents diagnoses for three different industrial examples. A plot of the collected data is shown in a figure as well as the diagnosis of the hypotheses.

The first data set was collected from an oscillating loop. The collected data can be seen in Figure 2.2.

The values calculated for the hypotheses can be seen in Table 2.1. The performance of the monitored control loop can be read out of the diagnoses. For this example hypotheses 1,2,3 and 11 are of interest. The conclusion of these hypotheses is that the loop is not oscillating due to valve problems. The problem lies either in the controller tuning or comes from an external disturbance.

Table 2.1: Diagnosis of oscillating example.

Hypothesis No.	**Hypothesis Name**	**Value**
1	**Sluggish Tuning**	**0**
2	**Loop Oscillating**	**1**
3	**Valve Problem**	**0**
4	Valve Leakage/Zero Error	-1
5	Setpoint Oscillating	0
6	Valve Travel	1
7	Noise	0
8	Acceptable Tracking	-1
9	Statistical behavior ok	0
10	Controller Saturated	0
11	**Acceptable Performance**	**0**

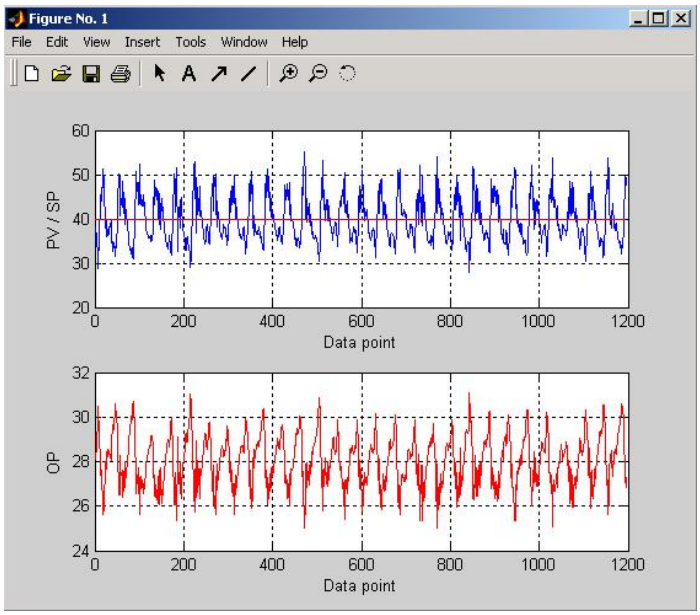


Figure 2.2: Oscillating control loop. Setpoint and process value(top) and controller output(bottom).

The second data set was collected from an ok behaving loop. The data can be seen in Figure 2.3

The values calculated for the hypotheses can be seen in Table 2.2. The hypothesis of interest is the last one, that says that the control loop is performing well, and no action is to be taken.

The next set of data is collected from a control loop that is in manual mode. Since the control signal is zero at all data points the signal is saturated as well. The data can be seen in Figure 2.4

The values given for the hypotheses can be seen in Table 2.3. The hypothesis of interest is number 10. This tells us that there is bad performance since the controller is "dead" $\equiv 0$. The control loop does not calculate any valve leakage, since it is not a flow loop.

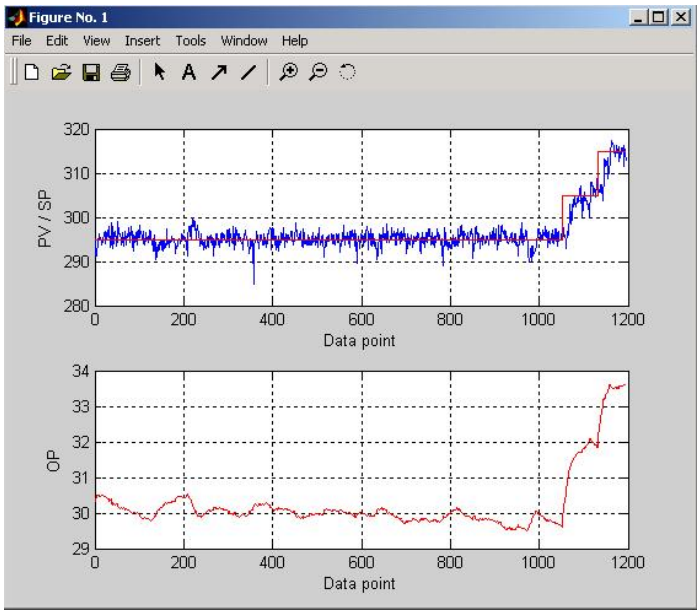


Figure 2.3: Well performing control loop. Setpoint and process value(top) and controller output(bottom).

Table 2.2: Diagnosis of ok example.

Hypothesis No.	**Hypothesis Name**	**Value**
1	Sluggish Tuning	0
2	Loop Oscillating	0
3	Valve Problem	-1
4	Valve Leakage/Zero Error	-1
5	Setpoint Oscillating	0
6	Valve Travel	0
7	Noise	1
8	Acceptable Tracking	-1
9	Statistical behavior ok	0
10	Controller Saturated	0
11	**Acceptable Performance**	**1**

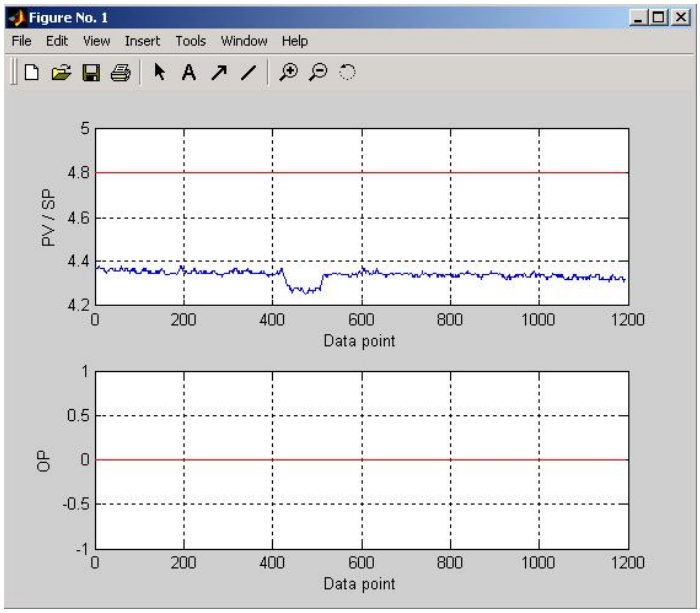


Figure 2.4: Control loop in manual mode. Setpoint and process value(top) and controller output(bottom).

Table 2.3: Diagnosis of saturated example.

Hypothesis No.	**Hypothesis Name**	**Value**
1	Sluggish Tuning	0
2	Loop Oscillating	0
3	Valve Problem	-1
4	Valve Leakage/Zero Error	-1
5	Setpoint Oscillating	0
6	Valve Travel	0
7	Noise	0
8	Acceptable Tracking	-1
9	Statistical behavior ok	0
10	**Controller Saturated**	**1**
11	Acceptable Performance	0

Chapter 3

Asset Optimization

3.1 General

The term *Asset Optimization* is widely used among companies these days. The meaning of the term and the goal of all companies, is to maximize the production and profit from the available resources. Having an asset working at peak performance, and more important keeping it there, is what is needed. Failures occur, and maintenance is at some point always needed, for all assets. It is at these moments most gains in production can be made. By minimizing the downtime of the asset and spotting an asset that is not working at peak performance as early as possible, much can be gained. To spot these assets, a lot of maintenance labour is wasted on routine checks, when no maintenance is needed as seen in Figure 3.1.

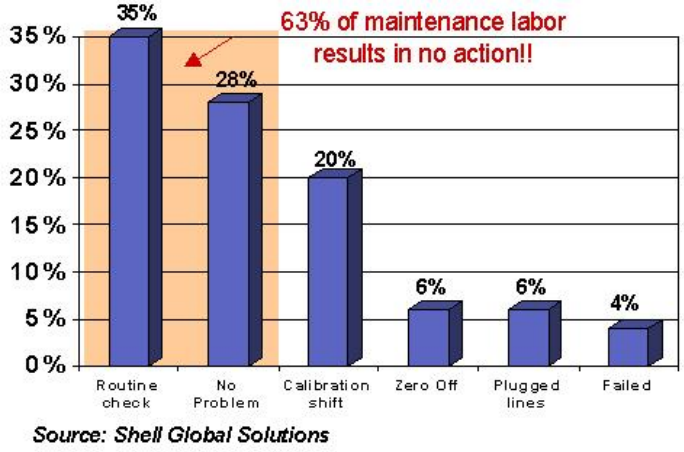


Figure 3.1: Maintenance labor usage.

If the asset that is not working at peak performance can be spotted without routine checks, they are not needed, and a lot of labour can be gained. What is needed is a system that spots these assets and the root cause of the problems automatically.

3.2 800xA Asset Optimization

ABB's state-of-the-art automation system is called **Industrial^{IT} Extended Automation System 800xA**. The system contains a single environment for engineering, operation and information management. 800xA collects, aggregates and analyzes real-time information from the whole plant and makes the necessary information available for the right people in real-time. The information flow in 800xA can be seen in Figure 3.2.

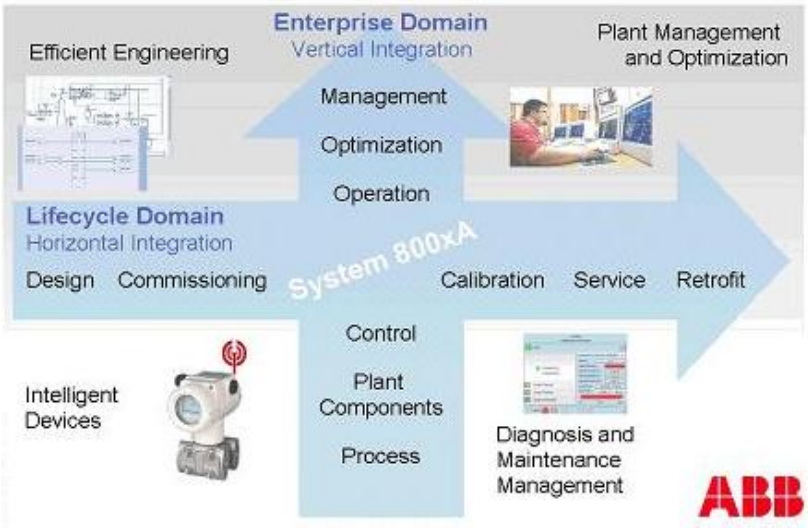


Figure 3.2: 800xA, platform for the complete integration of all data.

One of the core components within 800xA is called **Asset Optimization**. This component contains tools for tuning, improving or optimizing productive systems and end products as well as tools for monitoring control loops and other assets.

The plant explorer in 800xA is divided into three parts, as seen in Figure 3.3. The first part is the object browser. Here all objects in the current structure are displayed. Objects can be any kind of asset in the plant. The second part is the aspect browser. All aspects that the current object has, are displayed here. An aspect contains properties for the object. In the third part the details for the current aspect can be viewed and altered.

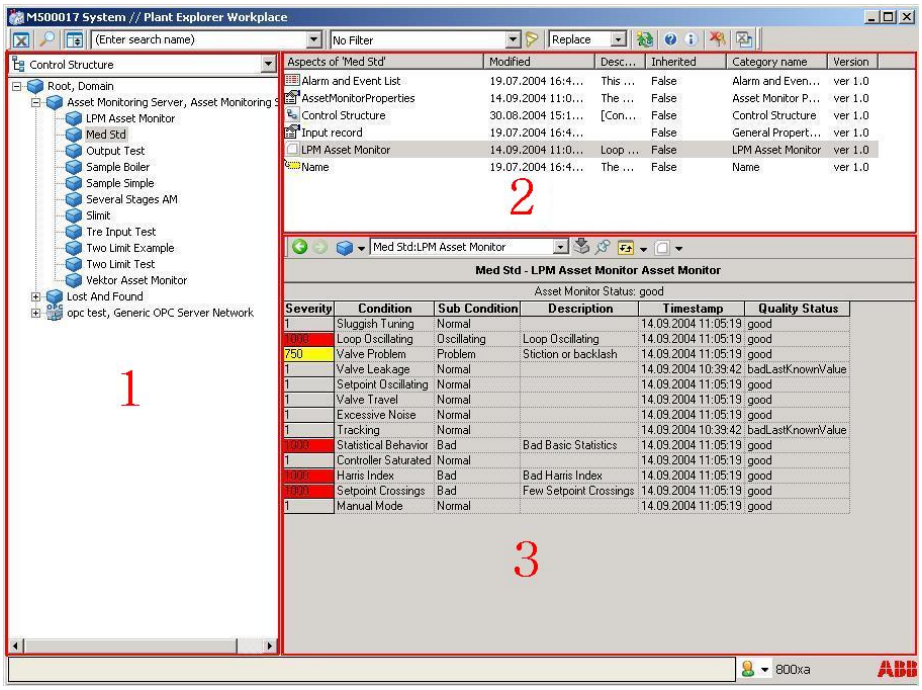


Figure 3.3: 800xA Plant Explorer.

3.3 Asset monitor

An object that constantly monitors the performance of an asset, is called an **Asset Monitor**. The object of this thesis is to implement an asset monitor for control loops in the 800xA system, and document how to do this. Typical objectives for an asset monitor are to [4]:

- Identify equipment failure indicators and implement predictive maintenance strategies.
- Identify performance degradation that may impact production output or quality.
- Identify efficiency degradation that may have economic or process impact.
- Identify faulty equipment.

The concrete implementation and architecture of an asset monitor is described in the next chapter.

Chapter 4

Control loop asset monitor

4.1 Asset monitor SDK

Several steps are needed to implement an asset monitor into the 800xA. They can be divided into four bigger steps as seen in Figure 4.1.

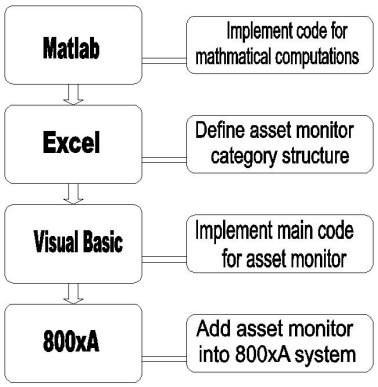


Figure 4.1: Implementation steps.

If the asset monitor has to perform complex computations, these should first be implemented in Matlab. This will make the main code implementation in Visual Basic easier later on. The next step is to create an asset monitor category, the structure for a whole category of assets. For example a boiler asset monitor category, which can monitor boilers. The asset monitor category is constructed through the **Asset Monitor Software Development Kit** (SDK) in **Microsoft Excel**. An asset monitor category could be something simple, like a high value check asset monitor category. That is an asset monitor category that checks if an input value is higher than a given limit value. All asset monitors in this category has the same structure, but they (could) have different input

signals and limit values. That means that the category only defines the structure of the asset monitor. All individual settings are then made in the 800xA system.

The structure of the control loop asset monitor should define how the performance and the state of the control loop are quantified. The asset monitor status is described through so-called conditions. Each condition can generate an alarm, the alarm depends on the subconditions that are defined for each condition. All conditions and subconditions have to be defined through the SDK. The number of inputs and outputs that the asset monitor category needs are also defined in the same Excel file.

The SDK enables the creation of a standard asset monitor form in Excel. This form contains several sheets, that describe what information the asset monitor category needs, and how it should be filled in. For example in one sheet the conditions and subconditions have to be defined. In another a file name, pointing to a visual basic program that should contain the logic for the asset monitor category, shall be defined. The SDK can then create the asset monitor category, making it available in the 800xA system.

Examples of exactly how to do this will be given later in this thesis.

4.2 Implementing the asset monitor

Once the structure of the asset monitor category is given, the logic of the asset monitor has to be implemented. This is done in **Visual Basic**. The basic structure of this logic is always the same. Among other things there must be an initialize function, that initializes variables, and an execute logic subroutine that implements the logic itself. In the execution logic, the variables' values that are defined in 800xA must be read and stored if they have been changed in 800xA. This could for example be the limit value in the high value check asset monitor category. Then the input records should be read, and their values be updated for the control loop asset monitor. The performance and status of the monitored control loop can then be calculated. If the calculations are complex or hard to implement in Visual Basic, they can be implemented as functions in e.g. Matlab, and then simply be called from the Visual Basic code. A call is then made to the 800xA updating the conditions and alarms there. A more detailed description will be given with the examples below.

4.3 Simple asset monitor example

Introduction

In this example the steps to implement a simple asset monitor in the 800xA system will be described in detail. This simple asset monitor will monitor if a control loop error is outside any of two different limits. First a simple Matlab function that does the mathematical computations will be developed. The conditions and input records for the asset monitor are defined in an Excel worksheet and uploaded automatically to the 800xA system through the asset monitor

SDK. A .dll file is then made out of a Visual Basic program, which contains the main logic. The Visual Basic code calls the Matlab function and uses the conditions and input records defined in the Excel worksheet. The asset monitor is then added into the 800xA system and the asset monitor can be tested. It is required that Matlab, Visual Basic, Microsoft Excel and a 800xA system are available on the used computer. The 800xA system must have the Optimize^{IT} Asset Optimization option and the Optimize^{IT} Asset Monitor SDK installed.

Matlab code

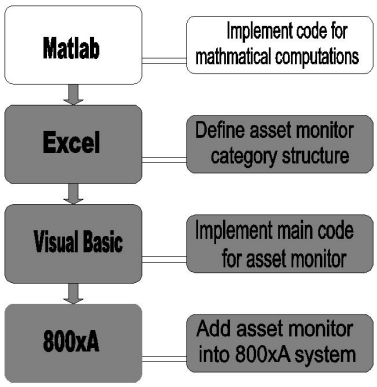


Figure 4.2: Step one: Matlab code.

The first step, as seen in Figure 4.2, is to implement the mathematical computations in Matlab. The Matlab function should in this example check if the absolute value sent to the function is larger than two limits provided to it. The return value of the function, `diagnosis`, should return an integer that should be zero if the input is smaller than both limits, one if the input is larger than the lower one of the limits and a two if the input is larger than the upper one of the limits. For example the code could be as follows, where `limit1` should be smaller than `limit2`:

checkLimits.m

```
function [diagnosis] = checkLimits(input, limit1, limit2)

   if abs(input) > limit2
      diagnosis = 2;
   elseif abs(input) > limit1
      diagnosis = 1;
   else
      diagnosis = 0;
   end
```

This code is not very advanced and could of course be implemented straight into Visual Basic. But the purpose here is only to show how to use Matlab functions in a Visual Basic program. This is useful for programs that have to compute advanced mathematical computations, which are easier to implement in Matlab.

To be able to use the Matlab function in Visual Basic the function has to be converted into a .dll file. The first step is to save the function, and to make sure that the path name to the file does not contain any blanks. If it does the Matlab function cannot be compiled into the .dll file. The compilation could also fail if the Matlab file contains some variable names that are forbidden, for example the variable name `Auto` will constrain the compilation to work. The Matlab COM Builder [5] is used, to create the .dll file (there are also other ways). This is accessed by writing: `comtool`, in the Matlab control window. The COM Builder graphical user interface (GUI) will then open, see Figure 4.3.

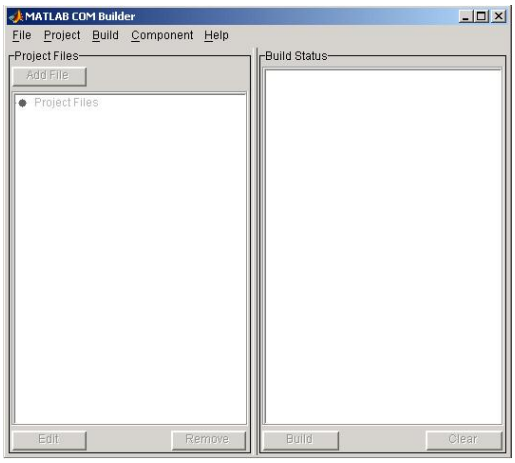


Figure 4.3: The COM Builder GUI.

Here the folder that should store the dll file will be created and the compi-

lation of the Matlab function into a .dll file will be made. Choose *File→New Project...* Here the component has to be given a name, for example *twoLimitExample*. Then click in the class name box. The rest of the fields will then be filled in automatically, as seen in Figure 4.4.

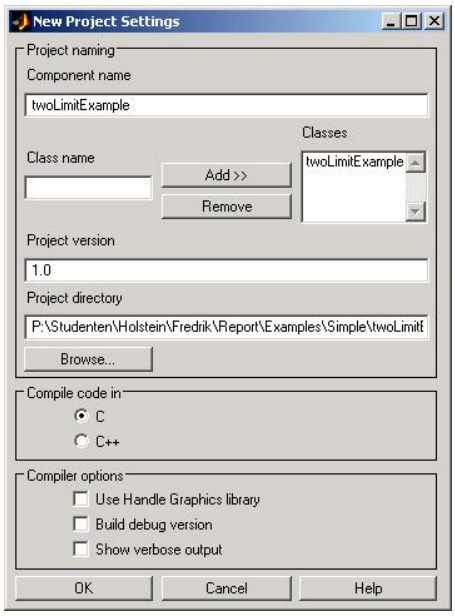


Figure 4.4: New Project.

Leave it like that and click *OK*. When asked weather to have this folder created, click *OK*. Click on the folder *twoLimitExample* in the COM Builder main window. Now the Matlab file has to be added to the project. This is done by clicking on the *Add File* button and choosing the *checkLimits.m* file. If several Matlab files wants to be added, they have to be added one at a time. When the file is added press the *Build* button. If no problems are encountered this will create the .dll file. Save the project and exit the program.

This completes the Matlab part of this example.

Asset monitor category

The steps to implement an asset monitor into the 800xA system are described in the **Asset Monitor SDK user's guide** [4]. Some steps are described more in detail there, and should be consulted for more information or if problems are encountered. But there are things that are not dealt with in the SDK manual that are described here.

The next step is to define the asset monitor category in Excel, as seen in Figure 4.5.

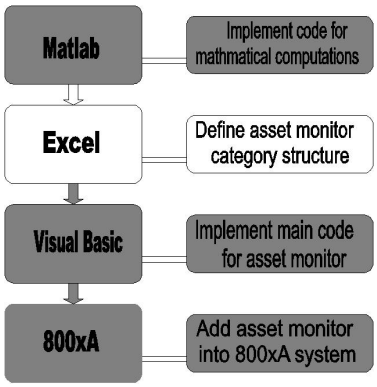


Figure 4.5: Step two: Asset monitor structure in Excel.

The conditions and input records (the signals from which the asset monitor should collect its data) for the asset monitor can conveniently be assigned in Microsoft Excel through the Asset Monitor SDK add-in. When Excel is opened there should be a toolbar with two buttons, *Update AM Category* and *Export AM Category*, in the Excel worksheet if the Optimize^{IT} Asset Monitor SDK is correctly installed. Open a new file and press the *Update AM Category* button and disregard the error messages. This will create the standard sheets that are needed to assign conditions and input records for the asset monitor category. Save the file, for example as *Two Limit Example Definitions*, and then press the *AM Definition* sheet. There the red cells have to be filled in with information about the asset monitor name and what the name of the asset monitor logic is. An example of how to fill them in is given in Figure 4.6.

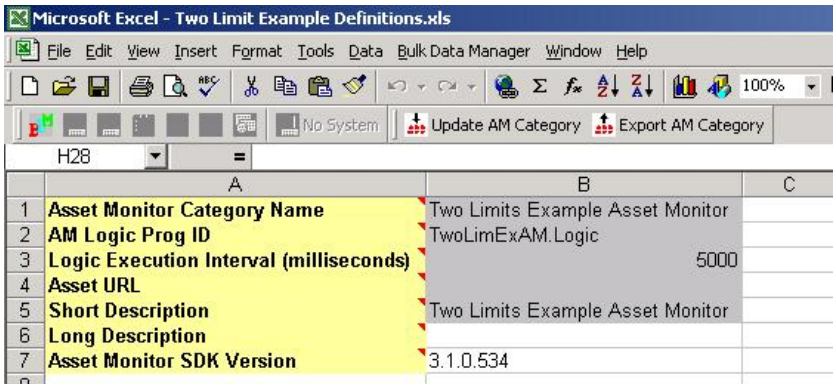


Figure 4.6: Asset Monitor definition.

The name of the asset monitor should describe the asset monitor, this will

make it easier to use and find later. The `AM Logic Prog ID` should be the name of the logic that will be implemented in Visual Basic later on, this should be remembered when the Visual Basic project is named. If the *Update AM Category* button is pressed again the red cells containing valid data will turn grey.

Next, click on the *Conditions* sheet. Here the different conditions and subconditions that should be monitored have to be defined. Every condition is something the asset monitor should observe and generate alarms for, for example if a value is over a limit. This means that each condition should have at least two subconditions. One of the subconditions should be the normal state, in which no alarm is active, this is where the control loop should be operating. The other subconditions generate different kind of alarms for the condition. For each subcondition a description of the alarm can be given, as well as a suggested action to correct it. Each subcondition should be given a different ENUM number (number that defines the different subconditions), this is used when the logic is implemented, to set the different subconditions. To measure how severe each alarm is, each sub condition is given a severity number. A higher number means that the alarm is more severe. The range of the severity number is 1-1000. The condition for this example can be seen in Figure 4.7.

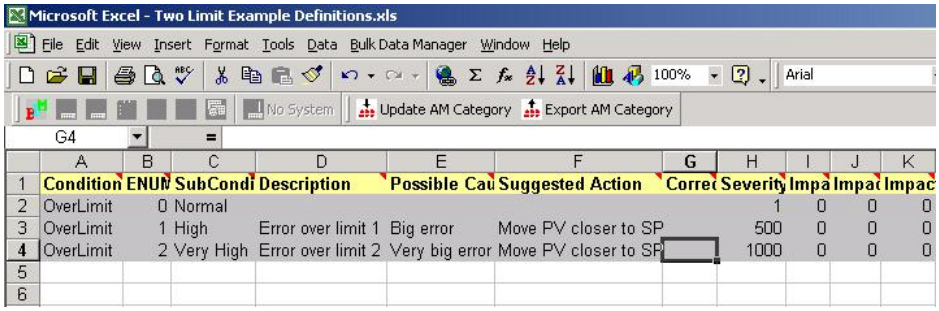


Figure 4.7: Conditions.

In this example there is only one condition with three subconditions. The condition can be named arbitrarily, here it is named `OverLimit`. The `ENUM` should contain the subcondition number. The zero has been reserved for the normal state. This state should also have one as severity number, since it is the normal state and no alarm should be active.

In the *Asset Parameters* sheet the two limits that are chosen should be filled in. All variables filled in here can have their values changed in the 800xA system later on. Suitable names for the parameters in this example are `Limit1` and `Limit2`. This can be seen in Figure 4.8.

Next thing to add into the Excel file is the input records. In this sheet the input signals, that the asset monitor should receive, have to be defined. Here three input signals are used, `INHIBIT`, `PV` and `SP`. The first input is used to

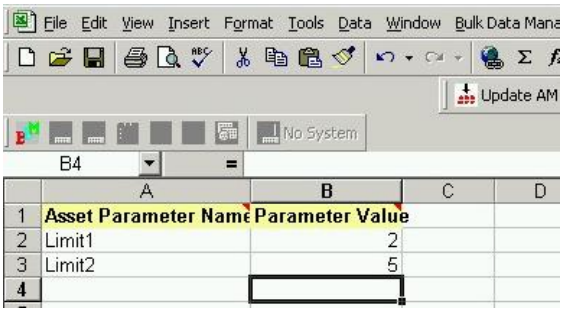


Figure 4.8: Asset Parameters.

inhibit the asset monitor from doing anything. This is used if the monitored asset is under maintenance or if there is temporary need to suppress alarms from the asset monitor. The other two should represent the process variable and the setpoint for a control loop, which are used to compute the control error for which the limit supervision is desired. See Figure 4.9 to see how to fill them in.

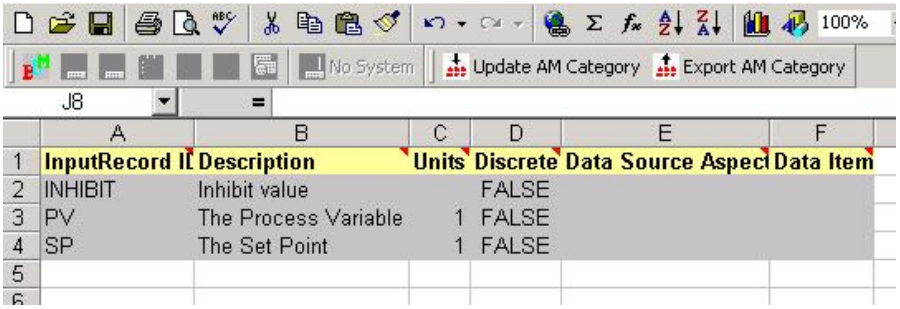
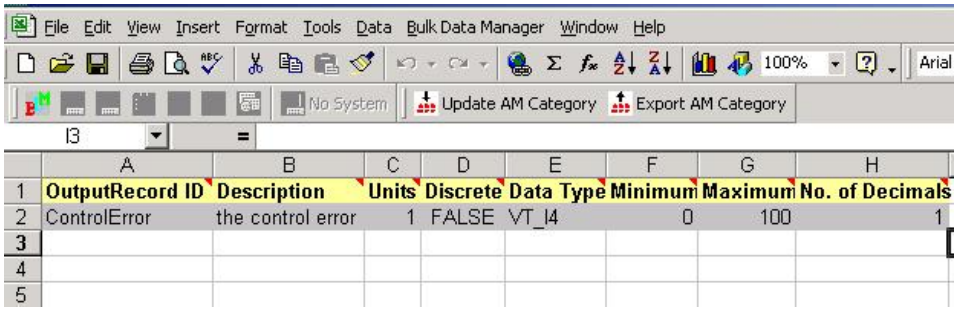


Figure 4.9: Input Records.

The last things to add are the output records. These are used to make values visible in the 800xA system. In this example the control error will be displayed. Supported output record data types are: VT_BSTR (string), VT_I4 (long), VT_R8 (double), VT_BOOL (boolean) and VT_DATE (date). An example of how to fill the fields is shown in Figure 4.10.

Now all the required definitions have been made. To upload the asset monitor category to the 800xA system press the *Export AM Category* button. Choose the `Two Limit Example Asset Monitor` and press *Export asset monitor into Excel*. Then press the *Update AM Category* button. This makes the `Two Limits Example Asset Monitor` available in the 800xA system. But there is no logic connected to the asset monitor. This has to be implemented in Visual Basic.



	A	B	C	D	E	F	G	H
1	**OutputRecord ID**	**Description**	**Units**	**Discrete**	**Data Type**	**Minimum**	**Maximum**	**No. of Decimals**
2	ControlError	the control error	1	FALSE	VT_I4	0	100	1
3								
4								
5								

Figure 4.10: Output Records.

Visual Basic

The logic of the asset monitor is implemented in Visual Basic. This is the next step in the asset monitor implementation, seen in Figure 4.11.

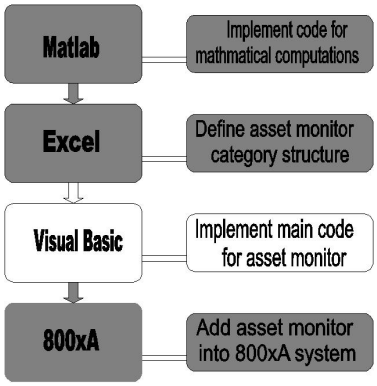


Figure 4.11: Step three: Main logic code in Visual Basic.

Start Visual Basic and choose to open a new *ActiveX DLL* project. Open the project properties. The project should here be given a suitable name, remember what logic name was given in the Excel work sheet. In this example the name should be `TwoLimExAM`. The property settings are shown in Figure 4.12.

The project also has to make some references. Open the project references (in the project menu). Here the following five references should be added, also seen in Figure 4.13.

- **AbbAoCommonUtils**
- **AbbAoMSservice**
- **AbbAoAmHelper**

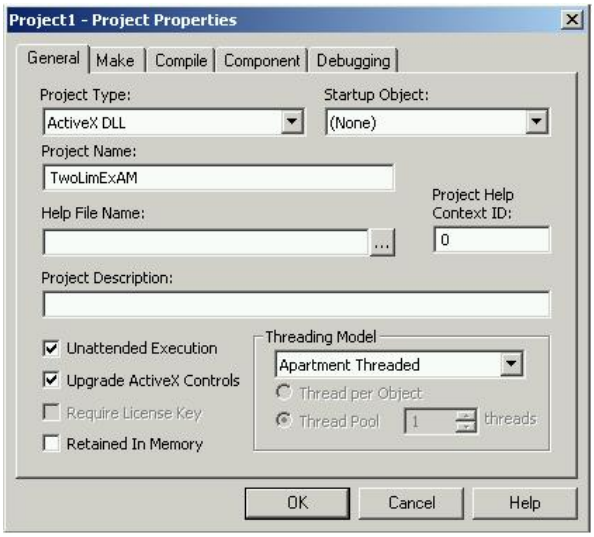


Figure 4.12: Visual Basic properties.

- **Microsoft XML, v3.0**
- **twoLimitExample 1.0 Type Library**

The first four references are needed to get the asset monitor working. The last reference is such that the Matlab function that was written can be found and used. The class name in the project then has to be changed to `Logic`. This can be done in the Visual Basic main view of the project.

Then create an original dll file by choosing *Make TwoLimExAM.dll* in the *File* menu. In the project properties under the *Component* tab then choose the *Binary Compability* option. If the file has not yet been saved, this should be done. Remember to have the class named *Logic*, and the project named the same as in the Excel file.

Now it is time to start implementing the main code. To be able to use all needed commands and functions the following has to be written in the header of the code:

```
Option Explicit
Option Compare Text
Implements AbbAoMSservice.ILogic
Private m_utils As AbbAoAmHelper.AssetMonitorUtils
Private lim As twoLimitExample.twoLimitExample
```

The first line always has to be written in Visual Basic when variables are to be declared. The next three lines are there such that the asset monitor and the

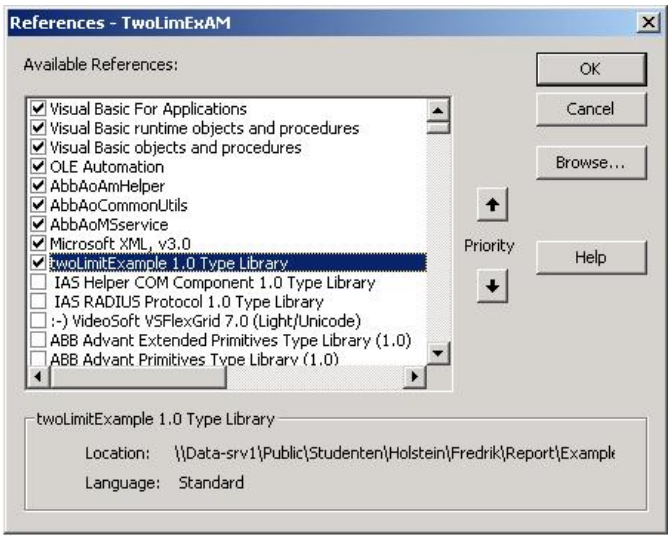


Figure 4.13: Visual Basic references.

dll made from the Visual Basic code can communicate. The last line declares a variable that is an instance of the Matlab dll file. The name of the two private variables can of course be different than above. Then the following property, function and subroutines have to be implemented in the code window in the Visual Basic project as seen in Figure 4.14.

- `Property Get Version() As String`
- `Function Initialize(ByRef Config As MSXML2.IXMLDOMNode) As String`
- `Sub ExecuteLogic(ByRef Assetmonitor_IN As AssetMonitor)`
- `Sub Terminate()`

The property *get version* is used to get the version of the implemented logic. This could be useful to keep track of different versions of the same logic out in the field. A typical implementation looks as:

```
Private Property Get ILogic_Version() As String
     ILogic_Version = App.Major & ".”  & App.Minor & ".”
                                                & App.Revision
End Property
```

The initialize function should initialize the private variables and return an empty string if this is successful. If any errors occur the error message should be returned, which is taken care of in the last part of the function. It will look

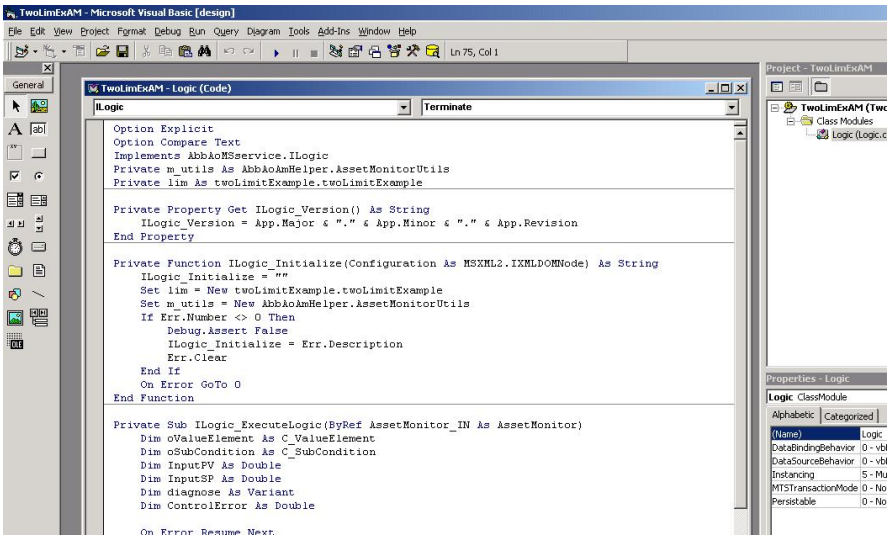


Figure 4.14: View of Visual Basic window.

as follows for this example:

```
Private Function ILogic_Initialize(Configuration
                                   As MSXML2.IXMLDOMNode) As String
      ILogic_Initalize = ” ”
      Set lim = New twoLimitExample.twoLimitExample
      Set m_utils = New AbbAoAmHelper.AssetMonitorUtils
      If Err.Number <> 0 Then
            Debug.Assert False
            ILogic_Initialize = Err.Description
            Err.Clear
      End If
      On Error GoTo 0
End Function
```

The terminate subroutine is quite simple and should only set the `m_utils` variable as nothing.

```
Private Sub ILogic_Terminate()
      Set m_utils = Nothing
End Sub
```

The last subroutine is the one containing the main logic. Here the limits and input records have to be read, the Matlab file should be called and the status

of the asset monitor should be updated. First the variables needed in the subroutine have to be declared. The beginning of the subroutine could be as follows.

```
Private Sub ILogic_ExecuteLogic(ByRef AssetMonitor_IN
                                              As Asset Monitor)
      Dim oValueElement As C_ValueElement
      Dim oSubCondition As C_SubCondition
      Dim InputPV As Double
      Dim InputSP As Double
      Dim diagnosis As Variant
      Dim ControlError as Double
```

The first variable is needed to get the values of the input records. The second one to set the subconditions. These two are in other words always needed. Next come the variables needed especially for this example. The diagnosis variable has to be of the *Variant* instance. The variables used in a call to a Matlab function have to be of the right instance, so that Matlab and Visual Basic can use the variable correctly. All variables that are returned from a call to a Matlab function have to be of the *Variant* instance. The variables sent in to the Matlab code should be of a suitable instance, depending on how it is used in the Matlab function. If an integer variable is sent into a Matlab file from Visual Basic, there can be no multiplications or divisions with this variable in the Matlab code, the call from Visual Basic to the Matlab function can in this case be unsuccessful and empty variables will be returned.

Next the limits have to be collected.

```
On Error Resume Next
With AssetMonitor_IN
      If .ConfigurationChanged Then
            .LogicBlockParameter1Desc = ”The first limit”
            .LogicBlockParameter1 = CDbl(
                .AssetParameters.selectSingeNode(”Limit1”).text)
            .LogicBlockParameter2Desc = ”The second limit”
            .LogicBlockParameter2 = CDbl(
                .AssetParameters.selectSingeNode(”Limit2”).text)
      End If
```

The first line is used in case an error occurs in the code when run. The second row is there so that `AssetMonitor_IN` does not have to be written every time it is called. It is enough to write a dot followed by the command in the `AssetMonitor_IN` that is wanted. The *if* line checks if the limits have been changed in 800xA, in this case they have to be loaded and stored again. The *if* line is also true the first time it is checked. There are 10 `LogicBlockParameters`, and they can contain any kind of variable, even vectors. These are used to store variable values for the individual asset monitors. It would not work with a public variable, since there can be many instances of the same asset monitor

category.

```
.InputRecords.Read
If m_utils.IsAssetMonitorInhibit(AssetMonitor_IN, "True",
                                                    "INHIBIT") Then
     Exit Sub
End If
.Status = m_utils.GetIORecordVE(.InputRecords,oValueElement,
                                                    ”PV”,NumericType)
If Len(.Status) = 0 Then
```

This piece of code first reads the input records. Then the `INHIBIT` value is checked. If the value is equal to the second parameter, in this case `True`, then all conditions will be set to the normal state and have the status *Inhibit* and the quality *goodLocalOverride*. After that the subroutine will be exited. If the asset monitor is not inhibited the next input record value will be read and if no errors occurs, `.Status` will be empty. The code should continue to run in this case. Inside the *if* case the next input record has to be read, after the first one has been stored in a local variable.

```
InputPV = oValueElement.Value
.Status = m_utils.GetIORecordVE(.InputRecords,oValueElement,
                                                    ”SP”,NumericType)
If Len(.Status) = 0 Then
```

After this the calculation part of the code can be executed.

```
InputSP = oValueElement.Value
ControlError = InputPV - InputSP

Call lim.checklimits(1, diagnosis, ControlError,
        .LogicBlockParameter1 , .LogicBlockParameter2 )

.StatusQuality = qualityStatusENUM.good
.Status = .Status & m_utils.WriteToOutputRecord(
     AssetMonitor_IN, ControlError, "ControlError", False)
Set oSubCondition = m_utils.SetCurrentSubCondition(
     AssetMonitor_IN, 1, diagnosis, False, .StatusQuality,
     .Status, False)
```

In this piece of code, the values for the variables are first stored. Then the call to the Matlab function is done. First, in the parameters given to the function, is the number of outputs, one in this case. Then the outputs should be given, they all have to be of the *Variant* instance. Last the inputs are given, in this case, the two limits. Once the call is successfully made, the quality of the conditions are given the value `good`. Then the output records are written

to. If that is done successfully, the `.Status` will be empty. Otherwise the error message is stored there. Last the subcondition is set to the value given in `diagnosis`. This is done in the `.SetCurrentSubCondition`. The place where `diagnosis` variable is placed is the place where the ENUM should be placed for the wanted subcondition, and this is conveniently the same value as in the `diagnosis`. If there is an error message in `.Status` , this will be shown in the description of the condition in 800xA.

Finally after the if cases have been ended the errors have to be taken care of.

```
            End If
      End If
If Err.Number <> 0 Then
      Debug.Assert False
      .Status = "Unhandled runtime error in
                  ILogic_ExecuteLogic(): "  & Err.Description
.StatusQuality = qualityStatusENUM.bad
Err.Clear
End If
End With
On Error GoTo 0
End Sub
```

The final thing to do is to make the dll file for the project once more, this has to be done every time a change has been made in the code.

That concludes the Visual Basic part of the asset monitor. An overview of the code can be seen in Appendix C.

800xA

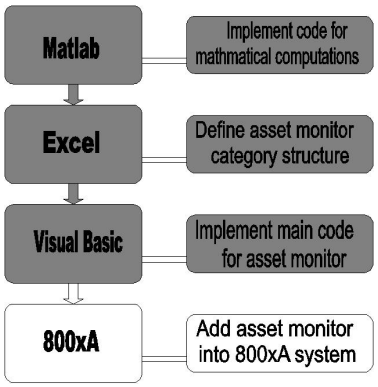


Figure 4.15: Step four: Asset monitor creation 800xA.

Now the asset monitor will be added into the 800xA system and tested. This is the last step in the asset monitor implementation, see Figure 4.15.

The asset monitor should be added in the control structure of the 800xA system. In the *Control Structure*, *Root*, add a new object to the *Asset Monitoring Server* as seen in Figure 4.16.

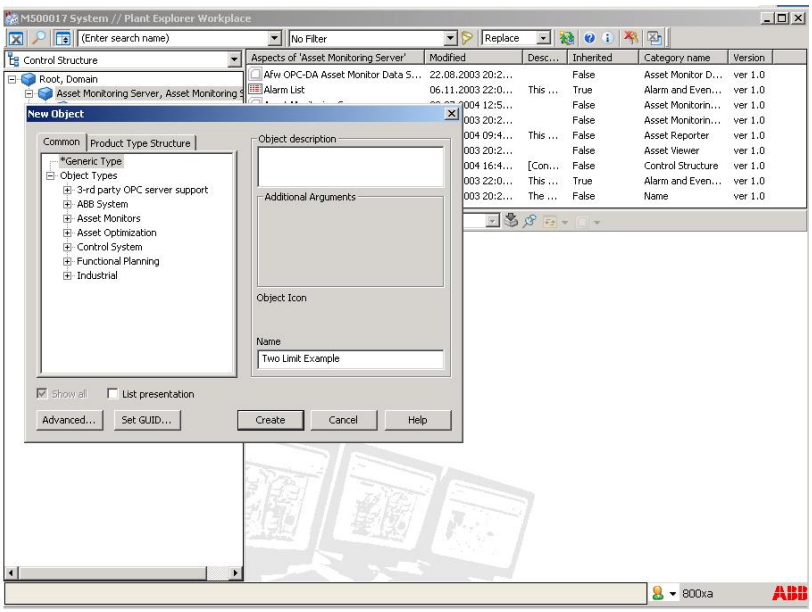


Figure 4.16: Adding a new object.

In this, object aspects then have to be added so that the asset monitor can function properly. First add a new *Alarm and Event List* aspect to the *Two Limit Example* object shown in Figure 4.17.

Next an aspect that generates input records has to be created. This aspect is needed to generate test data so that the asset monitor can be tested, and contains the value of the `INHIBIT` variable. Add this by choosing to add a new aspect, to the object, that is under *Basic Property Aspects*, *Basic Property Properties* and is called *General Properties*. Name the aspect `Input Records` as seen in Figure 4.18.

Last the asset monitor aspect has to be added. The *Two Limits Example Asset Monitor* can be found under *Asset Monitoring* and then *Asset Monitors*.

All the aspects that are needed have now been added. Now the aspects will be configured so the asset monitor can be tested. Go to the *Config View* of the *Two Limits Example Asset Monitor* as shown in Figure 4.19.

Under the *Server* tab set the *Monitoring Server* as `Default`. Then go to the *Input Records* tab and set all input records to be `Default` data source aspects. The *Data Source Item* is the path to where the asset monitor can find the input

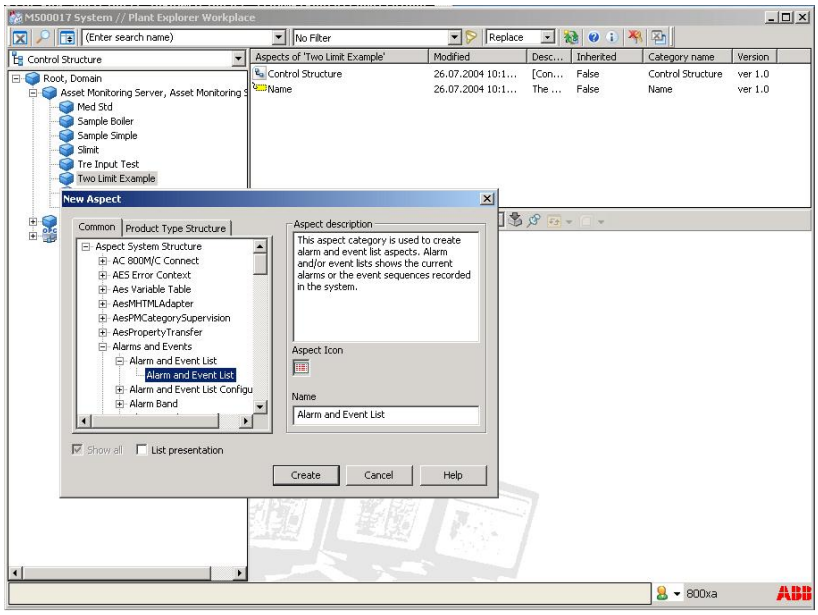


Figure 4.17: Adding a Alarm and Event List aspect.

records. To test the data we create input records in the *Input Records* aspect that was created earlier. The paths to these data are `%ID%:INHIBIT`, `%ID%:PV` and `%ID%:SP`. This can be seen in Figure 4.20. If the path is not known the *Open Properties Browser* button to the bottom right can be clicked and the path to the input records found by clicking through the folders. Click *Apply* to apply the new settings.

Next go to the *Config View* of the Input Records Aspect. Here the three input records, `INHIBIT`, `PV` and `SP`, have to be added. They should both be readable and writable, `PV` and `SP` should be of the data type `real`, and `INHIBIT` of the type `Boolean` (or `String`, but then the value of the variable has to be written manually every time it is changed). This can be seen in Figure 4.21.

After all the input records have been added, apply the settings by pressing the *Apply* button. Now the asset monitor is ready to be tested.

Testing the asset monitor

To test the asset monitor the asset monitoring server has to be started. Go to the *Config View* of the asset monitoring server seen in Figure 4.22

Press the *Download/Restart* button. Now the asset monitor should be running. To see if it is running properly, go to the *Main view* of the server and press the *Status* button. The status should be good, otherwise there is probably something wrong with the implemented asset monitor and it cannot get

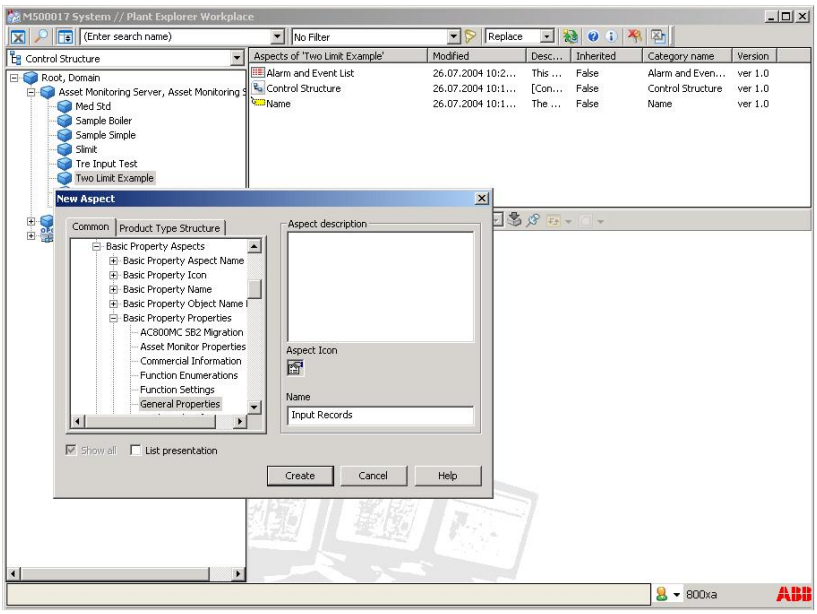


Figure 4.18: Adding a general properties aspect.

initiated. Once the server is up and running go to the input records and change the values so that the difference between them is larger than one of the limits that has been defined, remember to press the *Apply* button. Then go to the asset monitor aspect and the severity should change as seen in Figure 4.23 The output records can be seen by going to the *AssetMonitorProperties* aspect, seen in Figure 4.24. If the value of the `INHIBIT` input record is set to `True` the asset monitor should be inhibited.

If the alarms work as they should, a working asset monitor has been created. For more advanced asset monitors, the basics are still the same as in this example.

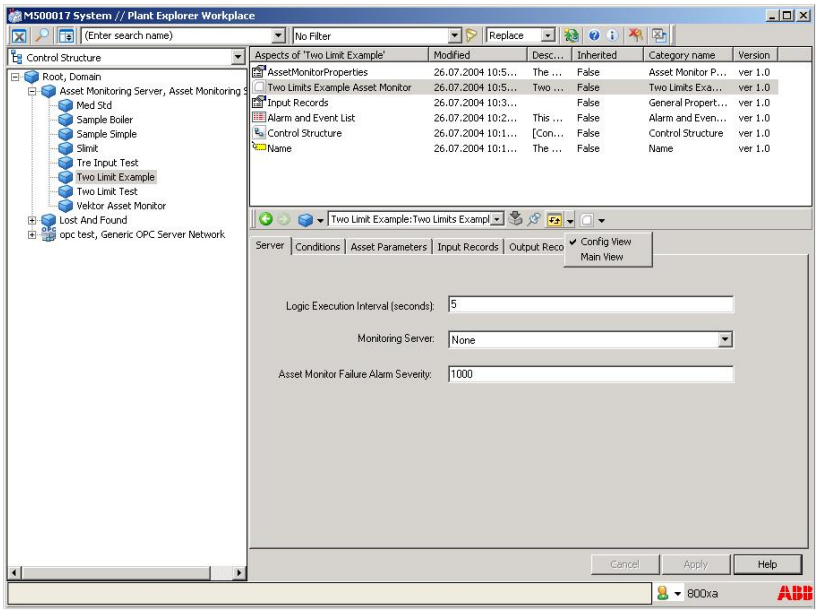


Figure 4.19: Config View of asset monitor.

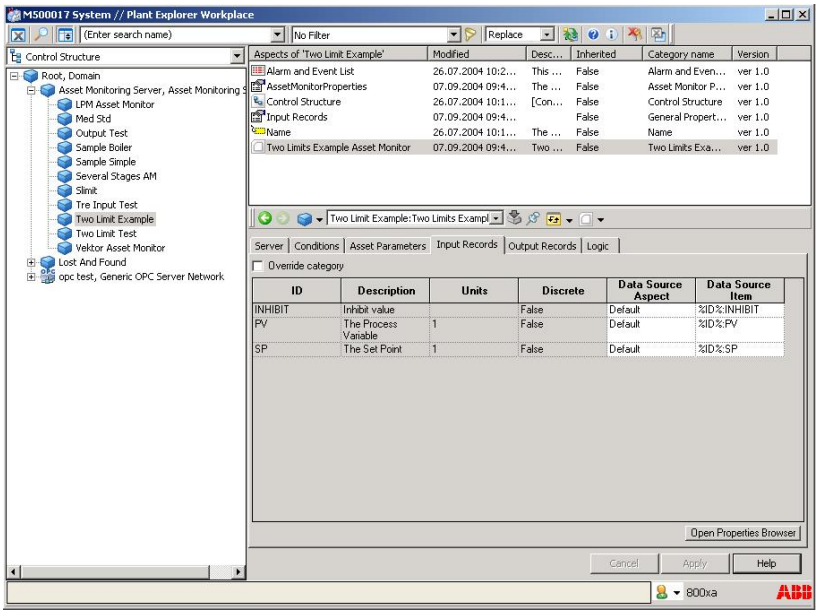


Figure 4.20: Configure path to input records.

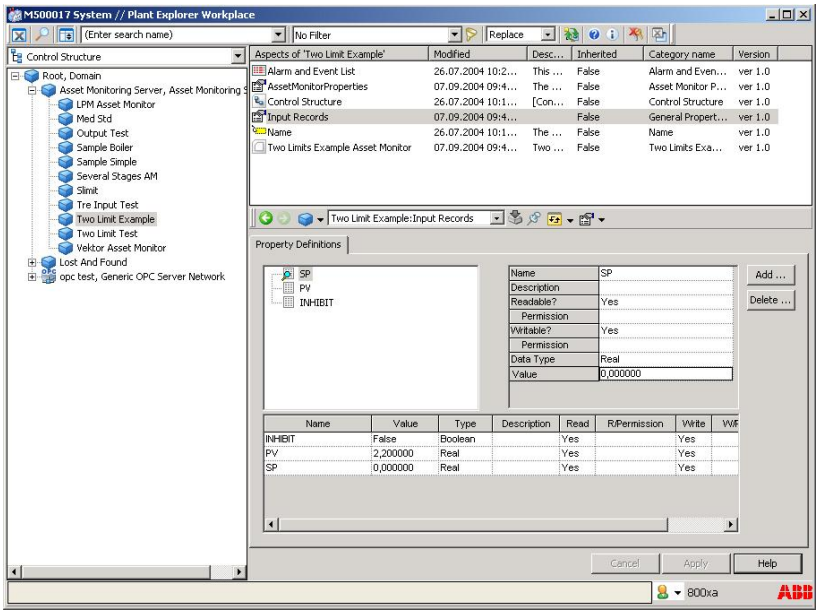


Figure 4.21: Set the input records.

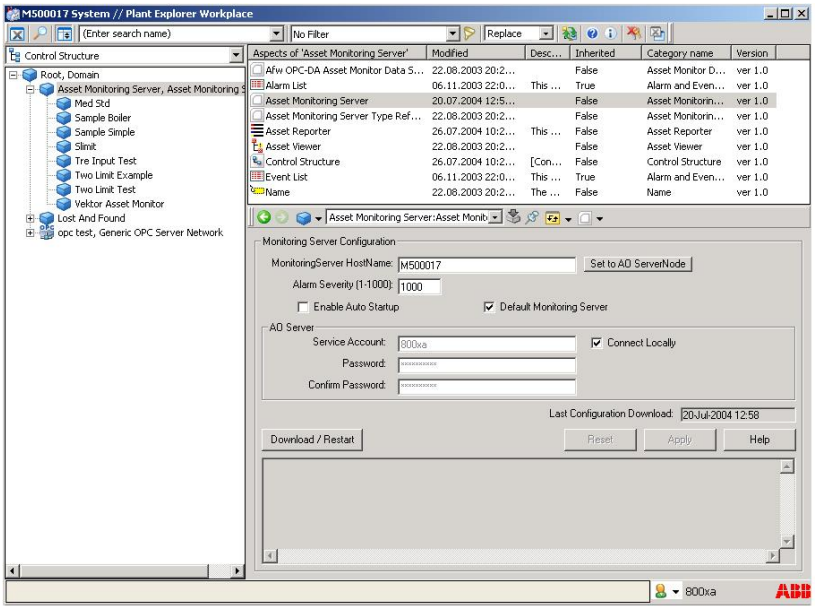


Figure 4.22: Starting the server.

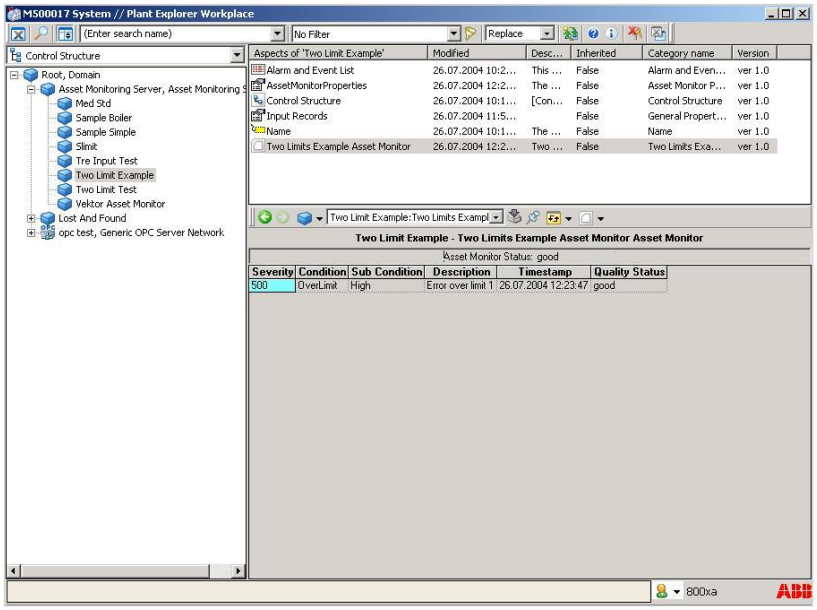


Figure 4.23: Testing the asset monitor.

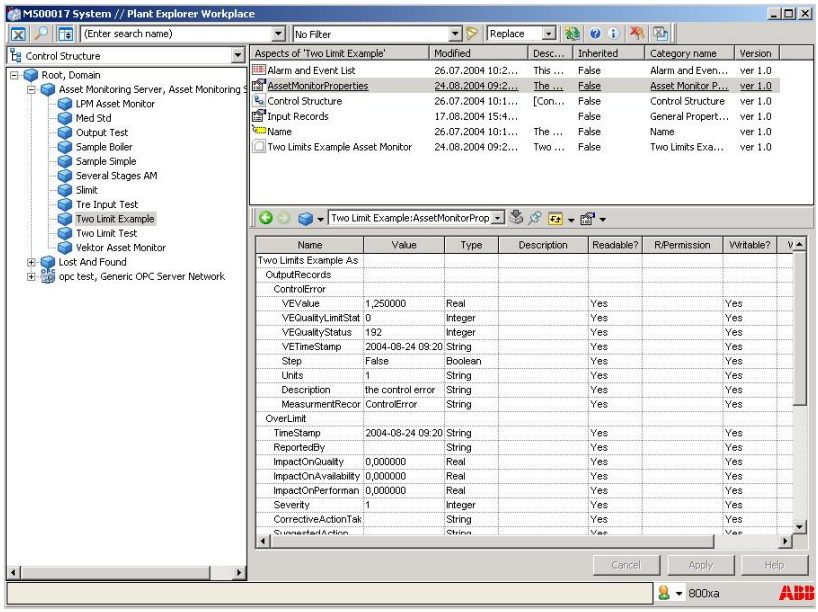


Figure 4.24: Output records in 800xA.

4.4 The LPM asset monitor

Introduction

This is the big example that is based on the Loop Performance Manager auditing visual software. The asset monitor uses the same Matlab functions as the LPM, but with a few modifications. This means that a batch of data has to be collected before the call to the Matlab functions can be made. So the alarms for the asset monitor will not be updated after every new sample that is collected. But it is not crucial for an asset monitor to inform about a problem the second it appears, most problems appear due to wear of the equipment and are spotted fast enough anyway. If more acute problems occur, these will be spotted through other means than asset monitors.

The basic steps to implement the asset monitor are the same as for the previous example, see Figure 4.1.

Matlab code

The Matlab code in this example is basically the same as implemented in LPM. There are two calls to Matlab functions from the Visual Basic code in this asset monitor. The first one is to `CLPA_Indices`, that calculates the indices from vectors containing control signal, process variable and set point values. The other call is made to `CLPA_Diagnosis`, that evaluates the hypotheses from the current indices. The inputs and outputs for the Matlab functions can be seen in Appendix D.

The Matlab functions have gone through some smaller changes to agree better with the asset monitor implementation in Visual Basic. Some code has been added, to make sure that the indices and hypotheses vectors are column vectors, ensuring that Visual Basic can recognise them as vectors. Changes have also been made in the diagnosis evaluation. Since the hypotheses sometimes are true for good performance and sometimes for bad, this was changed so that all hypotheses are true for bad performance. The last hypothesis was also divided into the different preconditions, some the same as some hypotheses, giving a total of 13 hypotheses. The valve problems hypothesis was also changed, it has two different levels of bad performance, so that a 2 is returned instead of 0.75 for the less certain bad performance. This was done so that two levels of the same alarm can be generated for this hypothesis in 800xA.

Asset monitor category

The asset monitor category is built up the same way as in the simple example, but with more parameters. The conditions for the LPM asset monitor can be seen in Figure 4.25.

The conditions are set so that the values returned from the diagnosis made from the Matlab code can be used directly when setting the conditions.

Some of the asset parameters given here are most for testing the asset monitor, when there is no real process to test it on. On a real process they could

	A	B	C	D	E	F	G	H
1	Condition ID	ENUM	SubCondition	Description	Possible Cause	Suggested Action	Correc	Sever
2	Sluggish Tuning	0	Normal					1
3	Sluggish Tuning	1	Bad	Sluggish Tuning	Controller tuning too slow	Retune controller		1000
4	Loop Oscillating	0	Normal					1
5	Loop Oscillating	1	Oscillating	Loop Oscillating	Bad controller tuning, actuator fault or external distubance	Scrutinize control loop		1000
6	Valve Problem	0	Normal					1
7	Valve Problem	1	Problem	Stiction or backlash	Fault within the actuator/valve assembly	Check actuator and valve		1000
8	Valve Problem	2	Problem	Stiction or backlash	Fault within the actuator/valve assembly	Check actuator and valve		750
9	Valve Leakage	0	Normal					1
10	Valve Leakage	1	Leak	Valve Leakage	Valve not tight shut-off or flowmeter zero is incorrectly set	Check valve and flowmeter		1000
11	Setpoint Oscillating	0	Normal					1
12	Setpoint Oscillating	1	Oscillating	Set Point Oscillating	Oscillating cascade master loop	Check master loop		1000
13	Valve Travel	0	Normal					1
14	Valve Travel	1	High	Excessive Movement	Tight controller tuning	Retune controller		1000
15	Excessive Noise	0	Normal					1
16	Excessive Noise	1	High	Excessive Measurment Noise	Controller tuning or sensor	Check controller and sensor		1000
17	Tracking	0	Normal					1
18	Tracking	1	Bad	Tracking Behavior Bad	Bad controller tuning	Retune controller		1000
19	Statistical Behavior	0	Normal					1
20	Statistical Behavior	1	Bad	Bad Basic Statistics	Bad controller tuning, quanitisation or non-linearity	Check controller and signal processing		1000
21	Controller Saturated	0	Normal					1
22	Controller Saturated	1	High	Loop Saturated	Bad controller tuning or bad dimensioning of actuator	Retune controller and check actuator		1000
23	Harris Index	0	Normal					1
24	Harris Index	1	Bad	Bad Harris Index	Bad controller tuning	Retune controller		1000
25	Setpoint Crossings	0	Normal					1
26	Setpoint Crossings	1	Bad	Few Setpoint Crossings	Bad controller tuning, actuator fault or external distubance	Scrutinize control loop		1000
27	Manual Mode	0	Normal					1
28	Manual Mode	1	High	Loop In Manual Mode	Controller set in manual mode	Check reason for manual mode		1000

Figure 4.25: Conditions for LPM asset monitor.

also be given as input records and read in automatically. The asset parameters can be seen in Figure 4.26.

	A	B
1	Asset Parameter Name	Parameter Value
2	LoopCategory	Flow_liquid
3	Cascade	slave
4	N	1000
5	Ts	1
6	OPmin	0
7	OPmax	100
8	LRmin	0
9	LRmax	100

Figure 4.26: Asset parameters for LPM asset monitor.

The variable N represents the size of the batch; it is the number of values stored in each vector before evaluation. To get all hypotheses evaluated correctly, there should be more than 200 data points. Ts is the sampling time. The OP limits are the limits for the control signal and the LR limits are the limits for the process variable and the set point.

The input records are the control signal, process variable, set point and an inhibit signal. These can be seen in Figure 4.27.

The inhibit signal is used when repairing the monitored control loop, or if for some reason the asset monitor should not collect data for a period of time. The logic implemented in visual basic should load the new input records, store them in the vectors and if they are full make a diagnosis of the asset monitor.

	A	B	C	D	E	F
1	InputRecord ID	Description	Units	Discrete	Data Source	Data Item
2	INHIBIT	Inhibit signal		TRUE	Default	%ID%:INHIBIT
3	U	Control signal	0,01	FALSE	default	
4	Y	Process variable	0,01	FALSE	default	
5	R	Set point	0,01	FALSE	default	

Figure 4.27: Input records for LPM asset monitor.

In the output records, most of the indices will be displayed, as seen in Figure 4.28.

	A	B	C	D	E	F	G	H
1	OutputRecord ID	Description	Units	Discrete	Data Ty	Minimun	Maxim	No.
2	mean_CE	control error, SP-PV	1	FALSE	VT_I4	-99999	99999	2
3	stdev_CE_norm	standard deviation of normalised control error	1	FALSE	VT_I4	0	100	2
4	mean_PV	mean process variable, current operating point	1	FALSE	VT_I4	0	1000	2
5	stdev_OP	standard deviation of controller output	1	FALSE	VT_I4	0	100	2
6	skew_CE_norm	measure of signal symmetry	1	FALSE	VT_I4	-99999	99999	2
7	kurt_CE_norm	measure of signal flatness	1	FALSE	VT_I4	-99999	99999	2
8	maxbicoher_CE	test data for nonlinearity, linear if <1.x	1	FALSE	VT_I4	0	99999	2
9	ratio_CE_OP	ratio of control error and controller output	1	FALSE	VT_I4	-99999	99999	2
10	mode_automatic	percentage of samples where loop in automatic	1	FALSE	VT_I4	0	100	2
11	mode_saturation	percentage of samples when loop saturated	1	FALSE	VT_I4	0	100	2
12	mode_cascade	percentage of samples when loop acts as slave	1	FALSE	VT_I4	0	100	2
13	shutoffvalue_PV	PV value if OP is saturated on lower limit	1	FALSE	VT_I4	0	1000	2
14	oscillation_CE	oscillation index control error	1	FALSE	VT_I4	0	1	2
15	osc_period_time	period [s] based on time-domain analysis	1	FALSE	VT_I4	0	99999	2
16	osc_period_freq	period [s] based on frequency-domain analysis	1	FALSE	VT_I4	0	99999	2
17	osc_amplit_time	amplitude based on time-domain analysis	1	FALSE	VT_I4	0	99999	2
18	osc_amplit_freq	amplitude based on frequency-domain analysis	1	FALSE	VT_I4	0	99999	2
19	osc_severity	percentage of varition contribution of main oscillation frequency	1	FALSE	VT_I4	0	100	2
20	flag_stiction1	stiction test based on cross-correlation method	1	FALSE	VT_I4	0	1	2
21	flag_stiction2	stiction test based on histogram method	1	FALSE	VT_I4	0	1	2
22	valve_travel	integrated valve movement per hour (with deadzone)	1	FALSE	VT_I4	0	99999	2
23	ACF_ratio_index	Harris-like index on ACF instead of closed loop impulse response	1	FALSE	VT_I4	0	500	2
24	crossing_SP	ratio # of setpoing crossings and all samples	1	FALSE	VT_I4	0	100	2
25	tracking_SP	ratio # of setpoint movements and all samples	1	FALSE	VT_I4	0	100	2
26	travel_ratio_SP_PV	ratio movement of SP and PV	1	FALSE	VT_I4	-99999	99999	2
27	control_performance	Harris index based on FCOR algorithm	1	FALSE	VT_I4	0	1	2
28	outlier	number of identified (not removed!!) outlier in promille	1	FALSE	VT_I4	0	1000	2

Figure 4.28: Output records for LPM asset monitor.

Visual Basic

The Visual Basic code has the same structure as the simple example. The get version, initialize and terminate functions are the same. Even the structure inside the execute logic subroutine is the same: check if configuration is changed, read input records, call Matlab functions and update status in 800xA. Since there are many values to update if the configuration is changed, this is made in its own subroutine, called *AssetMonitorConfigurationChanged*. Here the asset parameter values are read and stored in `.LogicBlockParameters`, the same for the batch vectors. Since there are more parameters that should be stored than there are `.LogicBlockParameters`, some of the parameters are stored in

a vector, that is stored in one of the `.LogicBlockParameters`. To ensure that variables are kept in their right form, some of the values will be sent back and forth between `.LogicBlockParameters` and local variables, here and later in the code.

Then the input records will be read and the current control signal, process variable and the set point will be stored in the batch vectors. If the batches are full, the calls to the Matlab functions are made, and new indices and hypotheses values are calculated. The index thresholds are set in a Matlab function that is called by the hypotheses calculation function.

The alarms will then be set, according to the hypotheses, in 800xA. The output records will be displayed through a call to a local subroutine. Their values are sent to a local variable, before the call to display the output record is made, to ensure that the variable is of the right instance.

The complete code can be seen in Appendix E.

800xA

As for the simple example, an object has to be added to the control structure of 800xA. The aspects for Alarms and Events and for the Input Records then have to be added. The asset monitor aspect that is added should be of the LPM Asset Monitor kind. After the right path for the input records have been set, and the asset parameters have their correct values, the asset monitor can be tested.

4.5 Testing the LPM Asset Monitor

Once the asset monitor was completely implemented into the 800xA system it was tested on the same test data as the LPM, see section 2.2.3. This was done by creating an OPC (Object linking and embedding for Process Control) client that reads the values from a Microsoft Excel file and sends them to an OPC server once every sampling time. The 800xA can then read the values from the server. If the asset monitor samples with too short sampling time it will read the same value for more than one sampling time. It was found that a sampling time of at least five seconds prevented this. This means that the server updates its values every five seconds and that the asset monitor reads new values every five seconds. If the length of the batches collected is the same as for the Matlab code in section 2.2.3, this would take too long time to collect for just testing. So the length of the batches was set to 250 samples instead. This does not change the result of the evaluations. But more samples usually make the indices more accurate. The lower limit for how many samples that are needed are among other things in the Harris index calculation. To be able to identify a good time series model, at least 200 samples are needed. The data collected in the batch for the first example, the oscillating control loop, can be seen in Figure 4.29.

And the alarms generated by the asset monitor in 800xA can be seen in Figure 4.30.

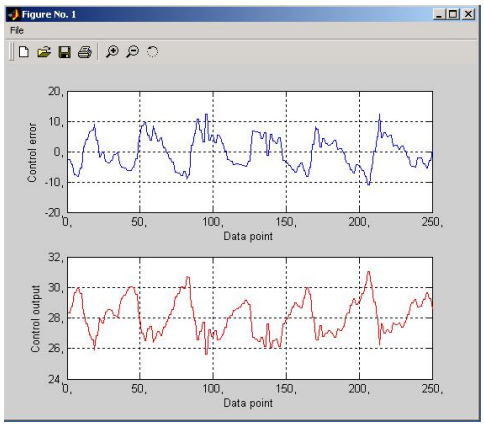
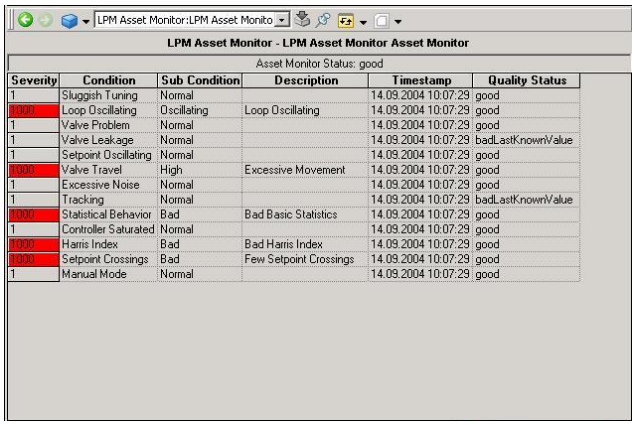


Figure 4.29: Oscillating control loop data. Setpoint and process value(top) and controller output(bottom).



Severity	Condition	Sub Condition	Description	Timestamp	Quality Status
1	Sluggish Tuning	Normal		14.09.2004 10:07:29	good
1000	Loop Oscillating	Oscillating	Loop Oscillating	14.09.2004 10:07:29	good
1	Valve Problem	Normal		14.09.2004 10:07:29	good
1	Valve Leakage	Normal		14.09.2004 10:07:29	badLastKnownValue
1	Setpoint Oscillating	Normal		14.09.2004 10:07:29	good
1000	Valve Travel	High	Excessive Movement	14.09.2004 10:07:29	good
1	Excessive Noise	Normal		14.09.2004 10:07:29	good
1	Tracking	Normal		14.09.2004 10:07:29	badLastKnownValue
1000	Statistical Behavior	Bad	Bad Basic Statistics	14.09.2004 10:07:29	good
1	Controller Saturated	Normal		14.09.2004 10:07:29	good
1000	Harris Index	Bad	Bad Harris Index	14.09.2004 10:07:29	good
1000	Setpoint Crossings	Bad	Few Setpoint Crossings	14.09.2004 10:07:29	good
1	Manual Mode	Normal		14.09.2004 10:07:29	good

Figure 4.30: Oscillating control loop alarms.

As can be seen in Figure 4.30, the result of the evaluation is the same as for the LPM. Which should be expected, since the asset monitor uses the same Matlab functions for its evaluations. Remember that some of the hypotheses have been changed, so that the performance is bad if they are true in the asset monitor.

For the next example, again only 250 data points were used. The data collected in the batch for the evaluation can be seen in Figure 4.31.

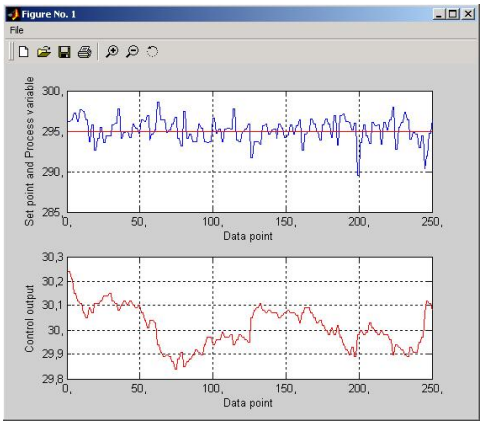
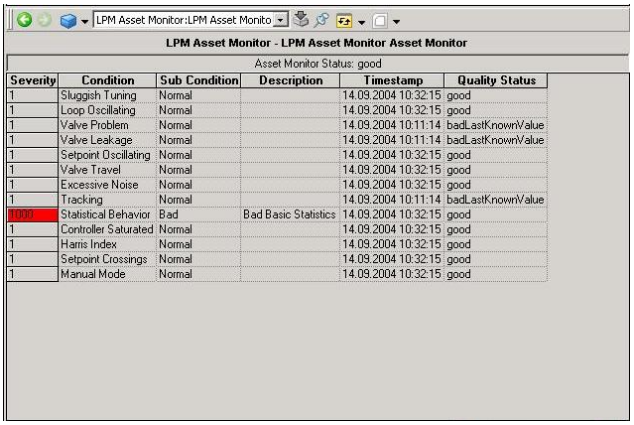


Figure 4.31: Ok control loop data. Setpoint and process value(top) and controller output(bottom).

And the alarms generated by the asset monitor in 800xA can be seen in Figure 4.32. All more severe alarms (e.g. Oscillation and Harris) are ok, but the controller can probably still be tuned better. See Figure 4.25, to see what the different alarms mean and examples of how to fix the problems. This can also be seen in 800xA, in the *Config view* of the asset monitor.



Severity	Condition	Sub Condition	Description	Timestamp	Quality Status
1	Sluggish Tuning	Normal		14.09.2004 10:32:15	good
1	Loop Oscillating	Normal		14.09.2004 10:32:15	good
1	Valve Problem	Normal		14.09.2004 10:11:14	badLastKnownValue
1	Valve Leakage	Normal		14.09.2004 10:11:14	badLastKnownValue
1	Setpoint Oscillating	Normal		14.09.2004 10:32:15	good
1	Valve Travel	Normal		14.09.2004 10:32:15	good
1	Excessive Noise	Normal		14.09.2004 10:32:15	good
1	Tracking	Normal		14.09.2004 10:11:14	badLastKnownValue
1000	Statistical Behavior	Bad	Bad Basic Statistics	14.09.2004 10:32:15	good
1	Controller Saturated	Normal		14.09.2004 10:32:15	good
1	Harris Index	Normal		14.09.2004 10:32:15	good
1	Setpoint Crossings	Normal		14.09.2004 10:32:15	good
1	Manual Mode	Normal		14.09.2004 10:32:15	good

Figure 4.32: Ok control loop alarms.

For the next example the same number of data points were used again. The data collected in the batch for the evaluation can be seen in Figure 4.33. The control signal is here turned off and will be viewed as saturated.

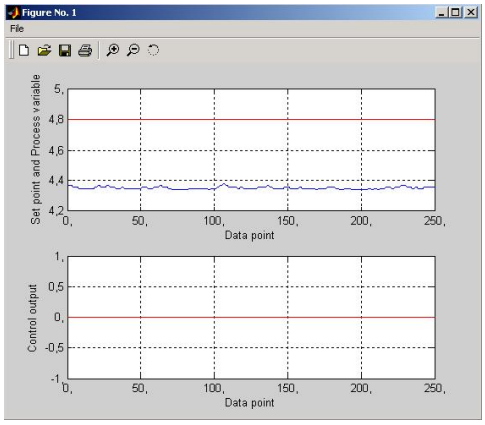
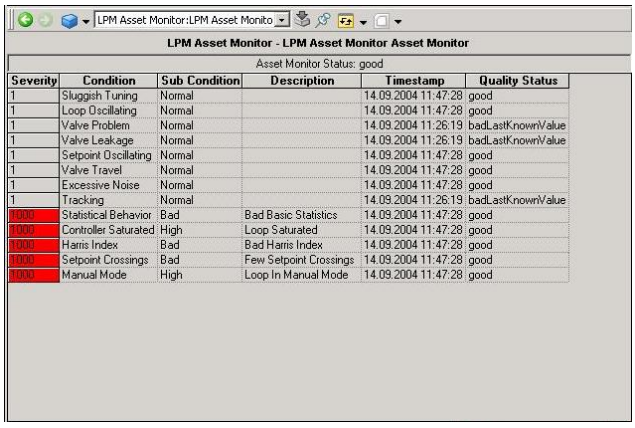


Figure 4.33: Saturated control loop data. Setpoint and process value(top) and controller output(bottom).

The alarms generated by the asset monitor in 800xA can be seen in Figure 4.34. If the control loop had been a flow loop, the valve leakage alarm would also have been raised.



Severity	Condition	Sub Condition	Description	Timestamp	Quality Status
1	Sluggish Tuning	Normal		14.09.2004 11:47:28	good
1	Loop Oscillating	Normal		14.09.2004 11:47:28	good
1	Valve Problem	Normal		14.09.2004 11:26:19	badLastKnownValue
1	Valve Leakage	Normal		14.09.2004 11:26:19	badLastKnownValue
1	Setpoint Oscillating	Normal		14.09.2004 11:47:28	good
1	Valve Travel	Normal		14.09.2004 11:47:28	good
1	Excessive Noise	Normal		14.09.2004 11:47:28	good
1	Tracking	Normal		14.09.2004 11:26:19	badLastKnownValue
1000	Statistical Behavior	Bad	Bad Basic Statistics	14.09.2004 11:47:28	good
1000	Controller Saturated	High	Loop Saturated	14.09.2004 11:47:28	good
1000	Harris Index	Bad	Bad Harris Index	14.09.2004 11:47:28	good
1000	Setpoint Crossings	Bad	Few Setpoint Crossings	14.09.2004 11:47:28	good
1000	Manual Mode	High	Loop In Manual Mode	14.09.2004 11:47:28	good

Figure 4.34: Saturated control loop alarms.

This concludes the stored data testing of the LPM Asset Monitor. The asset monitor is working as it should and can now be tested on live data. This is described in the next chapter.

Chapter 5

Applications

5.1 Simulink environment

One cause of oscillating loops could be stiction. It was decided to test the asset monitor for this kind of oscillations. Stiction is when the control valve is stuck in a certain position due to high static friction. When the control signal is large enough, the valve will start to move and then often it will move over to the other side of the desired set point, where it will get stuck again. This will then go on, making the control loop oscillating. Typical stiction data is when the control signal looks like a triangle wave, and the process variable signal looks more like a square wave.

A stiction model [6] was implemented in Matlab Simulink. Through an OPC server the values could be sent to 800xA, and the LPM asset monitor could be tested on this live data. The data batch that the asset monitor collected can be seen in Figure 5.1, where the typical stiction signals can be seen clearly.

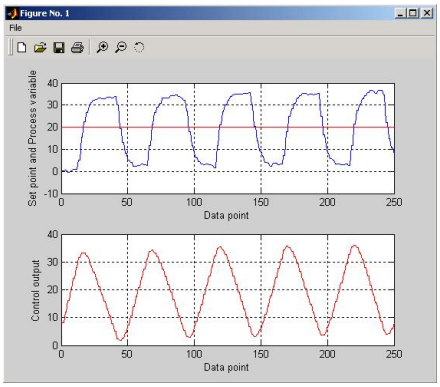
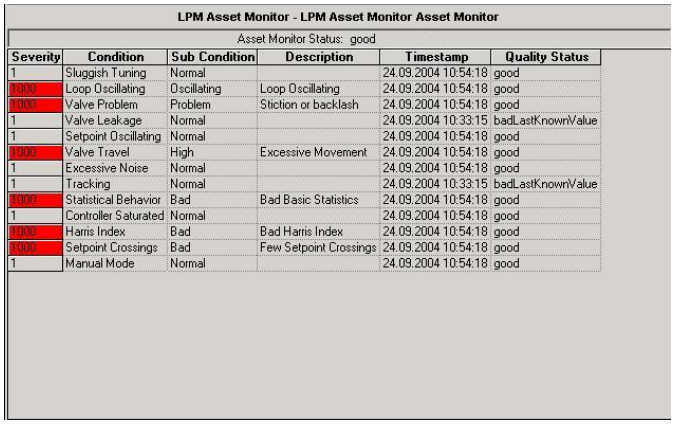


Figure 5.1: Data for stiction model. Setpoint and process value(top) and controller output(bottom).

The alarms generated in the asset monitor can be seen in Figure 5.2. The interesting alarms in this case are the oscillation and valve problems alarms. These indicate that the loop is oscillating, and that the cause of the problem could be stiction. This means that the actuator and the valve in the control loop have to be checked and maybe be reassembled or exchanged.



LPM Asset Monitor - LPM Asset Monitor Asset Monitor

Asset Monitor Status: good

Severity	Condition	Sub Condition	Description	Timestamp	Quality Status
1	Sluggish Tuning	Normal		24.09.2004 10:54:18	good
1000	Loop Oscillating	Oscillating	Loop Oscillating	24.09.2004 10:54:18	good
1000	Valve Problem	Problem	Stiction or backlash	24.09.2004 10:54:18	good
1	Valve Leakage	Normal		24.09.2004 10:33:15	badLastKnownValue
1	Setpoint Oscillating	Normal		24.09.2004 10:54:18	good
1000	Valve Travel	High	Excessive Movement	24.09.2004 10:54:18	good
1	Excessive Noise	Normal		24.09.2004 10:54:18	good
1	Tracking	Normal		24.09.2004 10:33:15	badLastKnownValue
1000	Statistical Behavior	Bad	Bad Basic Statistics	24.09.2004 10:54:18	good
1	Controller Saturated	Normal		24.09.2004 10:54:18	good
1000	Harris Index	Bad	Bad Harris Index	24.09.2004 10:54:18	good
1000	Setpoint Crossings	Bad	Few Setpoint Crossings	24.09.2004 10:54:18	good
1	Manual Mode	Normal		24.09.2004 10:54:18	good

Figure 5.2: Stiction alarms.

In the following example, the monitored control loop will be part of a real process with the industrial control system 800xA.

5.2 Lab process with industrial automation

The asset monitor was tested on a lab scale process with industrial components. The view of the process in 800xA can be seen in Figure 5.3 and a photo of the real process in Figure 5.4.

The big tank in the bottom supplies the two upper tanks with fluid. The flow from the bottom tank can be changed in the 800xA system. Both upper tanks have their levels measured and controlled. The level of the tank to the left is controlled by a control loop that changes the flow from the tank by opening and closing a valve. The input to the tank is seen as a disturbance and since only the level in the tank is measured and not the flows in or out, the process is very slow. The tank to the right has the same features as the other but it has an additional flow meter and a controller at the flow out from the tank. This control loop takes the desired output flow from the level control loop as its set point. It is on this control loop the asset monitor has been tested.

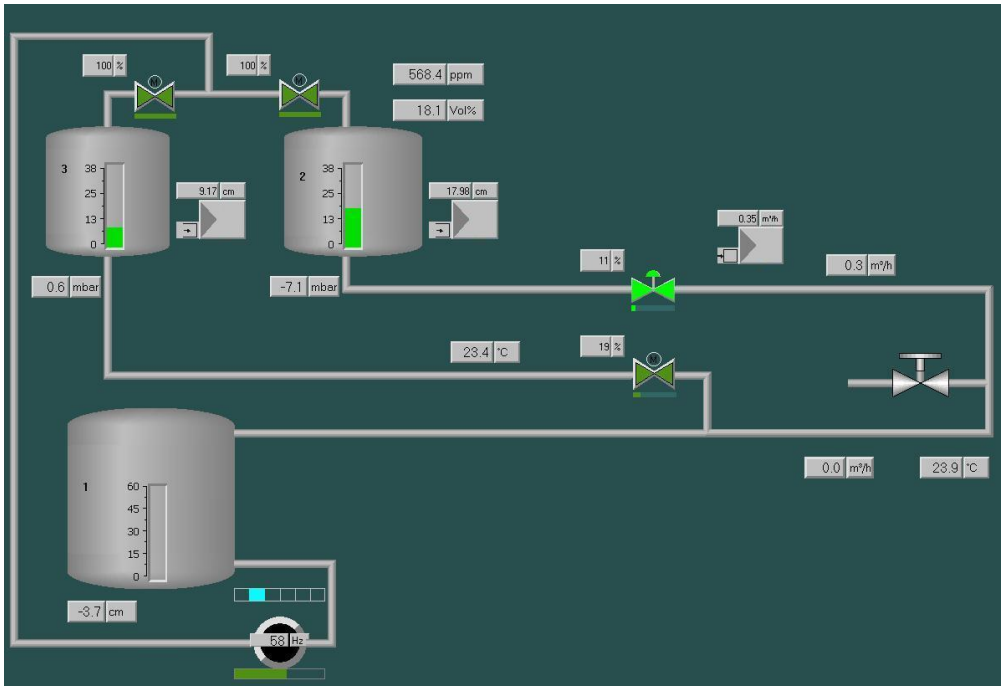


Figure 5.3: Lab process as seen in the 800xA plant explorer.

Figure 5.4: Lab process.

The values of the control input(OP), process variable(PV) and set point(SP) for a time period while the asset monitor was tested can be seen in Figure 5.5.

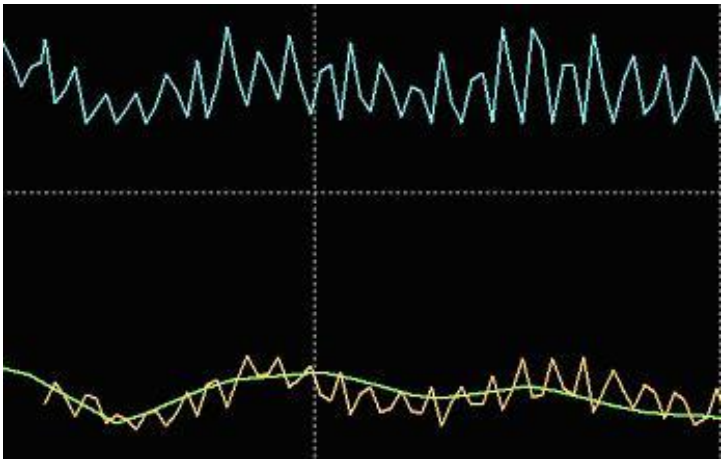
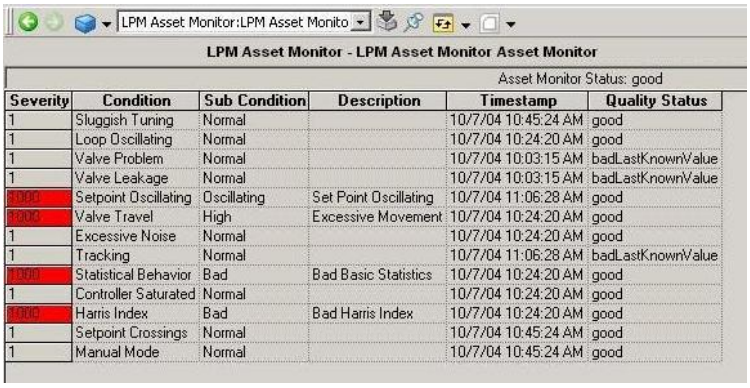


Figure 5.5: Lab process data: OP - blue (top), PV - yellow (fast osc) and SP - green (slow osc).

The time span for this picture is just over three minutes. When testing the asset monitor a sampling time of five seconds was chosen, for the same reasons as earlier when the asset monitor was tested, see Section 4.5. The small oscillations will then hardly be seen as oscillations since the oscillations are so small. But these oscillations mean that the controller has to work very hard to keep the process variable close to the set point, the control signal travels a lot. This can be seen in the alarms that are generated for the control loop seen in Figure 5.6.



LPM Asset Monitor:LPM Asset Monito

LPM Asset Monitor - LPM Asset Monitor Asset Monitor

Asset Monitor Status: good

Severity	Condition	Sub Condition	Description	Timestamp	Quality Status
1	Sluggish Tuning	Normal		10/7/04 10:45:24 AM	good
1	Loop Oscillating	Normal		10/7/04 10:24:20 AM	good
1	Valve Problem	Normal		10/7/04 10:03:15 AM	badLastKnownValue
1	Valve Leakage	Normal		10/7/04 10:03:15 AM	badLastKnownValue
1000	Setpoint Oscillating	Oscillating	Set Point Oscillating	10/7/04 11:06:28 AM	good
1000	Valve Travel	High	Excessive Movement	10/7/04 10:24:20 AM	good
1	Excessive Noise	Normal		10/7/04 10:24:20 AM	good
1	Tracking	Normal		10/7/04 11:06:28 AM	badLastKnownValue
1000	Statistical Behavior	Bad	Bad Basic Statistics	10/7/04 10:24:20 AM	good
1	Controller Saturated	Normal		10/7/04 10:24:20 AM	good
1000	Harris Index	Bad	Bad Harris Index	10/7/04 10:24:20 AM	good
1	Setpoint Crossings	Normal		10/7/04 10:45:24 AM	good
1	Manual Mode	Normal		10/7/04 10:45:24 AM	good

Figure 5.6: Lab process alarms.

There is an alarm generated for the slow oscillation that the set point has. This means that the level control loop is oscillating. The oscillation is due to that the process is slow and it takes to long time to get the desired out flow, and by the time the desired out flow is reached a new is already wanted. The two other alarms suggest that the flow controller can be tuned better.

To get a clearer picture of how the control loop is performing, the level controller was turned off, and the flow controller was fed with a constant set point. Data from this test can be seen in Figure 5.7.

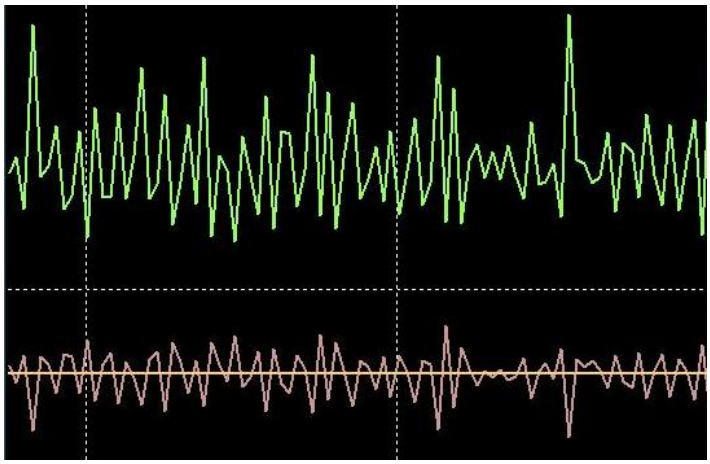


Figure 5.7: Lab process data: OP - green (top), PV - pink (lower moving) and SP - orange (constant).

Again the control signal seems to move a lot and the process variable does not settle to a constant value. It looks like the controller could be tuned better. A look at the alarms that were generated confirm this, as seen in Figure 5.8, confirms this. Both alarms that are generated suggest that the controller tuning should be reconfigured.

The main reasons for this test was to import the asset monitor into another 800xA system than it was developed on and test run it on data from a process with real industrial automation. This was successful and the asset monitor is ready for tests on real automation systems out in the industry.

LPM Asset Monitor - LPM Asset Monitor Asset Monitor

Asset Monitor Status: good

Severity	Condition	Sub Condition	Description	Timestamp	Quality Status
1	Sluggish Tuning	Normal		10/11/04 12:14:21 PM	good
1	Loop Oscillating	Normal		10/11/04 11:53:18 AM	good
1	Valve Problem	Normal		10/07/04 10:03:15 AM	badLastKnownValue
1	Valve Leakage	Normal		10/07/04 10:03:15 AM	badLastKnownValue
1	Setpoint Oscillating	Normal		10/11/04 11:53:18 AM	good
1000	Valve Travel	High	Excessive Movement	10/11/04 11:53:18 AM	good
1	Excessive Noise	Normal		10/11/04 11:53:18 AM	good
1	Tracking	Normal		10/07/04 11:06:28 AM	badLastKnownValue
1000	Statistical Behavior	Bad	Bad Basic Statistics	10/11/04 11:53:18 AM	good
1	Controller Saturated	Normal		10/11/04 11:53:18 AM	good
1	Harris Index	Normal		10/11/04 11:53:18 AM	good
1	Setpoint Crossings	Normal		10/11/04 12:14:21 PM	good
1	Manual Mode	Normal		10/11/04 12:14:21 PM	good

Figure 5.8: Lab process alarms.

Chapter 6

Final conclusions

The main purpose of this thesis was to implement a prototype asset monitor for control loops in ABB's automation system 800xA. To be able to understand the purpose and function of the asset monitor, control performance monitoring, asset optimization and the 800xA system were described and explained. The developed asset monitor is based on the Loop Performance Manager from ABB, so to be able to understand how the asset monitor is suppose to work this tool has to be understood. The LPM is given a batch of collected data. From this data several index values are calculated. Each index is a measure of how a certain part of the control loop is performing. The index values are then compared to preset limits. Depending on how the indices relate to the limits a diagnosis of the control loop is made. The diagnosis is described through hypotheses, statements, that can be true, false or unknown. This means, that given a diagnosis it can for example be said about the loop that it is oscillating, it is not saturated and so on.

The developed asset monitor works in the same manner. But it collects its own data, and when enough data (the amount set by the user) is colleted a diagnosis is made. The asset monitor is operating continuously, and as soon as something in a diagnosis is bad, events will be generated in the 800xA system with hints of what could be the cause of the problem.

The asset monitor was first tested on offline data, which the LPM also had been tested on. The results from the asset monitor were compared with the results from the LPM, and it was seen that the asset monitor worked as it should. It was then tested on online data, first on a control loop simulated in Matlab Simulink and then on a lab scale process. The asset monitor generated the alarms that were expected and worked as it should.

The SDK user manual[4] gave basic instructions of how to implement an asset monitor into 800xA. But at some points this manual is inferior. Much of the code for an asset monitor had to be generated through trial and error, since the code is not explained satisfactory in the manual. There were also problems to get the Matlab functions to operate as they should when called from the Visual Basic code. A lot of testing with different kind of variables

sent back and forth had to be done before the LPM Asset Monitor worked as it should. To facilitate further improvements and asset monitor implementations an implementation guide is also included in the thesis. This guide describes the basic steps to implement an asset monitor, into the 800xA system, in detail.

The next step for the asset monitor would be to test it in a real plant, on control loops that are operating in a real factory. For real industrial use the asset monitor has to be configured, so that all individual asset monitor settings are made automatically when the asset monitor is added to the desired object. This have to made manually as the asset monitor is now.

Improvements on the asset monitor functionality can be made in many areas, and among them are:

- more events can be added, for other control loop functionalities.
- each event can be divided to several levels with different severities.
- the algorithms that calculate the index values can be extended.
- new real time algorithms can be developed, so that the asset monitor can update the alarms for every new data sample that is collected.

These improvements are just examples for this control loop asset monitor, but also asset monitors in general have great potential for further research and development.

Appendix A

Indices for LPM

On the following page the details for all 38 indices in the LPM are displayed.

Index	index class	unit	min	max	nominal	lower limit	upper limit	brief description
mean loop error	basic statistics	real	n.a.	n.a.	0	n.a.	n.a.	control error, SP-PV
std loop error (norm)	basic statistics	%	0	100%	0	0	5	standard deviation of normalised control error
std loop error (eng.unit)	basic statistics	eng.unit	LRmin	LRmax	0	0	n.a.	standard deviation of control error in engineering units
mean PV	basic statistics	eng.unit	LRmin	LRmax	n.a.	n.a.	n.a.	mean process variable, current operating point
std OP	basic statistics	%	0	100%	n.a.	n.a.	n.a.	standard deviation of controller output
skewness control error	basic statistics	real	n.a.	n.a.	0	-1	1	measure of signal symmetry
kurtosis control error	basic statistics	real	n.a.	n.a.	0	-1.5	1.5	measure of signal flatness
maximum bicoherence	basic statistics	real	0	inf	0	0	1.5	test data for nonlinearity, linear if <1.x
gaussianity test	basic statistics	prob.	0	1	1	0.9	1	Gaussianity test using higher-order statistics
linearity test	basic statistics	real	n.a.	n.a.	1	0.5	2	Linearity test, test statistics is ratio
ratio CE / OP	modes	real	-inf	inf	n.a.	n.a.	n.a.	ratio of control error and controller output
automatic mode	modes	%	0	100%	100	0	10	percentage of samples where loop in automatic
saturation	modes	%	0	100%	0	0	2	percentage of samples when loop saturated
cascade mode	modes	%	0	100%	n.a.	0	10	percentage of samples when loop acts as slave
master flag	flags	[0/1]	0	1	n.a.	n.a.	n.a.	flag is loop is master or not (slave / inner loop in cascade). Single loops should be classified as slave, i.e. master flag = 0.
step flag	flags	[0/1]	0	1	n.a.	n.a.	n.a.	flag that indicates whether single setpoint steps were found. Such data is potentially valuable for plant model identification.
shut-off PV value	modes	eng.unit	LRmin	LRmax	n.a.	n.a.	n.a.	PV value if OP is saturated on lower limit
oscillation index CE	oscillation	real	0	1	n.a.	0	0.3	oscillation index control error
oscillation index SP	oscillation	real	0	1	n.a.	0	0.3	oscillation index setpoint
oscillation period (time)	oscillation	[s]	0	inf	n.a.	n.a.	n.a.	period [s] based on time-domain analysis
oscillation period (freq.)	oscillation	[s]	0	inf	n.a.	n.a.	n.a.	period [s] based on frequency-domain analysis
oscillation amplitude (time)	oscillation	%	0	inf	n.a.	n.a.	n.a.	amplitude based on time-domain analysis
oscillation amplitude (freq.)	oscillation	%	0	inf	n.a.	n.a.	n.a.	amplitude based on frequency-domain analysis
oscillation severity	oscillation	%	0	100	n.a.	0	25	percentage of varition contribution of main oscillation frequency
stiction (correlation)	oscillation	flag	0	1	n.a.	n.a.	n.a.	stiction test based on cross-correlation method
stiction (histogram)	oscillation	flag	0	1	n.a.	n.a.	n.a.	stiction test based on histogram method
noise index	modes	%	0	100%	n.a.	n.a.	n.a.	percentage of direction changes in PV
valve travel per hour	modes		0	inf	n.a.	n.a.	n.a.	integrated valve movement per hour (with deadzone)
ACF/pred.horizon ratio	perf. index	real	0	500	1	0	2	Harris-like index on ACF instead of closed loop impulse response
estimated time-delay	perf. index	real	0	inf	n.a.	n.a.	n.a.	estimation of time-delay
model fit flag	perf. index	flag	0	1	n.a.	n.a.	n.a.	flag if data is suitable for plant model identification
setpoint crossing index	modes	%	0	100%	25	15	100	ratio # of setpoing crossings and all samples
cascade tracking index	modes	%	0	100%	0	20	100	ratio # of setpoint movements and all samples
cascade travel ratio	modes	real	-inf	inf	1	0.5	1.5	ratio movement of SP and PV
Harris index	perf. index	real	0	1	1	0.6	1	Harris index
prediction horizon	parameter	samples	1	inf	n.a.	n.a.	n.a.	default values for each loop category, stored for documentation only
loop category	parameter	integer	1	9	n.a.	n.a.	n.a.	number which represents the loop categories
outlier	basic statistics	o/oo	0	1000	0	0	1	number of identified (not removed!!) outlier in promille

Figure A.1: Indices in the LPM.

Appendix B

Hypotheses for LPM

Here all the hypotheses and preconditions for the LPM are displayed.

Hypothesis number	name
1	sluggish tuning
2	loop oscillating
3	valve problem
4	valve leakage / zero error
5	setpoint oscillating
6	valve travel
7	noise
8	acceptable tracking
9	statistical behaviour
10	controller saturated
11	acceptable performance
Preconditions for 'acceptable performance'	
1	Harris index good
2	setpoint crossings index large
3	loop not oscillating
4	loop not saturated
5	loop not in manual
6	tracking acceptable
7	no valve leakage
8	no sluggish tuning

Figure B.1: Hypotheses in the LPM.

Appendix C

Visual Basic code Simple example

```
Option Explicit
Option Compare Text
Implements AbbAoMSservice.ILogic
Private m_utils As AbbAoAmHelper.AssetMonitorUtils
Private lim As twoLimitExample.twoLimitExample


Private Property Get ILogic_Version() As String
    ILogic_Version = App.Major & "." & App.Minor & "." & App.Revision
End Property


Private Function ILogic_Initialize(Configuration As MSXML2.IXMLDOMNode) As String
    ILogic_Initialize = ""
    Set lim = New twoLimitExample.twoLimitExample
    Set m_utils = New AbbAoAmHelper.AssetMonitorUtils
    If Err.Number <> 0 Then
        Debug.Assert False
        ILogic_Initialize = Err.Description
        Err.Clear
    End If
    On Error GoTo 0
End Function


Private Sub ILogic_ExecuteLogic(ByRef AssetMonitor_IN As AssetMonitor)
    Dim oValueElement As C_ValueElement
```

```
    Dim oSubCondition As C_SubCondition
    Dim InputPV As Double
    Dim InputSP As Double
    Dim diagnose As Variant
    Dim ControlError As Double

    On Error Resume Next
    With AssetMonitor_IN

        If .ConfigurationChanged Then
            .LogicBlockParameter1Desc = "The first limit"
            .LogicBlockParameter1 = CDbl(.AssetParameters.selectSingleNode
("Limit1").Text)
            .LogicBlockParameter2Desc = "The second limit"
            .LogicBlockParameter2 = CDbl(.AssetParameters.selectSingleNode
("Limit2").Text)
            Call .ConfigurationChangedAck
        End If

        .InputRecords.Read

        If m_utils.IsAssetMonitorInhibit(AssetMonitor_IN, "True", "INHIBIT") Then
            Exit Sub 'If inhibit then exit
        End If

        .Status = m_utils.GetIORecordVE(.InputRecords, oValueElement, "PV",
NumericType)
        If Len(.Status) = 0 Then

            InputPV = oValueElement.Value
            .Status = m_utils.GetIORecordVE(.InputRecords, oValueElement, "SP",
NumericType)
            If Len(.Status) = 0 Then

                InputSP = oValueElement.Value
                ControlError = InputPV - InputSP

                Call lim.checklimits(1, diagnose, ControlError,
.LogicBlockParameter1, .LogicBlockParameter2)

                .StatusQuality = qualityStatusENUM.good
                .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
ControlError, "ControlError", False)
                Set oSubCondition = m_utils.SetCurrentSubCondition(AssetMonitor_IN,
1, diagnose, False, .StatusQuality, .Status, False)
            End If
```

```
        End If

        If Err.Number <> 0 Then
            Debug.Assert False
            .Status = "Unhandled runtime error in ILogic_ExecuteLogic():
" & Err.Description
            .StatusQuality = qualityStatusENUM.bad
            Err.Clear
        End If
    End With
    On Error GoTo 0
End Sub


Private Sub ILogic_Terminate()
    Set m_utils = Nothing
End Sub
```

Appendix D

Matlab functions in the LPM Asset Monitor

CLPA_Indeices.m

```
[Indices, err, warn, IndexNames] = CLPA_Indices(y, r, u, ylow, yhigh,
ulow, uhigh, Ts, Looptype, cascade)
```

Where

`Indices`	The calculated indices for the current batch of data.
`err`	An array with error information.
`warn`	An array with warning information.
`IndexNames`	String containing the short names of all indices.
`y`	Vector containing all process values for the current batch.
`r`	Vector containing all controller set points for the current batch.
`u`	Vector containing all controller out puts for the current batch.
`ylow/yhigh`	Loop low/high range.
`ulow/uhigh`	Controller output limits.
`Ts`	Sample Time of log in seconds.
`Looptype`	Loop category for monitored control loop.
`cascade`	Cascade type ('master', 'slave').

CLPA_Diagnosis.m

```
[H,err,warn] = CLPA_Diagnosis(Indices)
```

Where

`H`	The evaluations of the hypotheses.
`err`	An array with error information.
`warn`	An array with warning information.
`Indices`	The calculated indices for the current batch of data.

Appendix E

Visual Basic code LPM Asset Monitor

```
Option Explicit
Option Compare Text
Implements AbbAoMSservice.ILogic ' Asset Monitor Logic must implement this interface
Private m_utils As AbbAoAmHelper.AssetMonitorUtils  ' Asset Monitor Helper functions
Private lim As LPMfunctions.matlabFunctions         ' Matlab functions


Private Property Get ILogic_Version() As String
' Returns the Version for this Logic.
    ILogic_Version = App.Major & "." & App.Minor & "." & App.Revision
End Property


Private Function ILogic_Initialize(Configuration As MSXML2.IXMLDOMNode) As String
' Called once upon startup to Initialize the Asset Monitor Logic
' Must Return "" if successful.
    ILogic_Initialize = ""

    Set lim = New LPMfunctions.matlabFunctions
    Set m_utils = New AbbAoAmHelper.AssetMonitorUtils

    If Err.Number <> 0 Then
        Debug.Assert False
        ILogic_Initialize = Err.Description
        Err.Clear
    End If
    On Error GoTo 0
```

```
End Function


Private Sub ILogic_ExecuteLogic(ByRef AssetMonitor_IN As AssetMonitor)
    Dim oValueElement As C_ValueElement ' used to get input values
    Dim InputRecordY As Double  ' stores current process variable value
    Dim InputRecordR As Double  ' stores current set point value
    Dim InputRecordU As Double  ' stores current control signal value
    Dim oSubCondition As C_SubCondition ' used to set sub conditions
    Dim hypotheses As Variant   ' returned hypothesis from matlab function,
                                ' must be variant
    Dim errors As Variant       ' returned errors from matlab function,
                                ' must be variant
    Dim warn As Variant         ' returned warnings from matlab function,
                                ' must be variant
    Dim names As Variant        ' returned index names from matlab function,
                                ' must be variant
    Dim i As Integer            ' used for For loops
    Dim vectorSize As Integer   ' stores the size of vectors
    Dim returnedIndices As Variant' returned indices from matlab function,
                                ' must be variant
    Dim y() As Double           ' process variable batch
    Dim r() As Double           ' set point batch
    Dim u() As Double           ' control signal batch
    Dim TempVector() As Double  ' used to store vectors betweeen variables,
                                ' needed since the LBP are of the Variant instance

    On Error Resume Next

    With AssetMonitor_IN
        ' Checks if Asset Monitor Configuration has changed
        If .ConfigurationChanged Then
            Call AssetMonitorConfigurationChanged(AssetMonitor_IN)
        End If

' if any problems were encountered a description
        ' is stored in .LogicBlockParameter9
        .Status = .LogicBlockParameter9

        If Len(.Status) > 0 Then
            ' Bad Configuration
            .StatusQuality = qualityStatusENUM.badConfigurationError
            ' Sets all the Asset Monitor Conditions to
            ' Normal - badConfigurationError quality and EXIT.
            Call m_utils.SetAllCurrentSubConditions(AssetMonitor_IN,
1, True, .StatusQuality, .Status, False)
```

```
            Exit Sub
        End If

        .InputRecords.Read                  ' read all input records

        '**** Inhibit Logic ****************************************************
        ' The following is the Standard inhibit logic that should be implemented in
' every Asset Monitor.
        ' LogicBlockParameter2 contains the value of the
        ' INHIBIT_VALUE Asset Parameter.
        If m_utils.IsAssetMonitorInhibit(AssetMonitor_IN, "True", "INHIBIT") Then
            Exit Sub 'If inhibit then exit
        End If

        '**** Store Values *****************************************************
        .Status = m_utils.GetIORecordVE(.InputRecords, oValueElement, "U",
NumericType)
        If Len(.Status) = 0 Then         ' no problems found if .status is empty
            InputRecordU = oValueElement.Value
            .Status = m_utils.GetIORecordVE(.InputRecords, oValueElement, "Y",
NumericType)
            If Len(.Status) = 0 Then     ' no problems found if .status is empty
                InputRecordY = oValueElement.Value
                .Status = m_utils.GetIORecordVE(.InputRecords, oValueElement, "R",
NumericType)
                If Len(.Status) = 0 Then ' no problems found if .status is empty
                    InputRecordR = oValueElement.Value

                    ' store values in batches if all input records are read with
                    ' no problems,
                    ' need to move values betweeen variables since LBP are of the
                    ' variant instance
                    y = .LogicBlockParameter7
                    r = .LogicBlockParameter8
                    u = .LogicBlockParameter6

                    y(.LogicBlockParameter3) = InputRecordY ' LBP3=position in batch
                    .LogicBlockParameter7 = y
                    r(.LogicBlockParameter3) = InputRecordR ' LBP3=position in batch
                    .LogicBlockParameter8 = r
                    u(.LogicBlockParameter3) = InputRecordU ' LBP3=position in batch
                    .LogicBlockParameter6 = u

                    ' if batches are full then.LBP3 = position,LBP2(6)=size of batch
                    If .LogicBlockParameter3 = .LogicBlockParameter2(6) Then
                    '**** Make Diagnosis ***************************************
```

```
                        ' calculate indices
                        Call lim.clpa_indices(4, returnedIndices, errors, warn, names,
.LogicBlockParameter7, .LogicBlockParameter8, .LogicBlockParameter6,
.LogicBlockParameter2(4), .LogicBlockParameter2(5), .LogicBlockParameter2(2),
.LogicBlockParameter2(3), .LogicBlockParameter2(1), .LogicBlockParameter1(1),
.LogicBlockParameter1(2))

                        TempVector = .LogicBlockParameter4
                        ' store values in vector
                        If IsArray(returnedIndices) Then
                            vectorSize = UBound(returnedIndices)
                            For i = 1 To vectorSize
                                TempVector(i) = returnedIndices(i, 1)
                            Next
                        End If

                        ' store now indices
                        .LogicBlockParameter4 = TempVector

                        ' make diagnosis
                        Call lim.clpa_diagnosis(3, hypotheses, errors, warn,
TempVector)

                        TempVector = .LogicBlockParameter5
                        ' store values in vector
                        If IsArray(hypotheses) Then
                            vectorSize = UBound(hypotheses)
                            For i = 1 To vectorSize
                                TempVector(i) = hypotheses(i, 1)
                            Next
                        End If

                        ' store new hypotheses
                        .LogicBlockParameter5 = TempVector

                        ' reset values
                        .LogicBlockParameter3 = 0
                        ReDim y(1 To .LogicBlockParameter2(6))
                        ReDim r(1 To .LogicBlockParameter2(6))
                        ReDim u(1 To .LogicBlockParameter2(6))

                        .LogicBlockParameter6 = u
                        .LogicBlockParameter7 = y
                        .LogicBlockParameter8 = r
```

```
                    End If

                    ' next position, LBP3 = position in batch
                    .LogicBlockParameter3 = .LogicBlockParameter3 + 1
                End If
            End If
        End If

        '**** Set Conditions ***************************************************
        .StatusQuality = qualityStatusENUM.good
        .Status = ""
        ' set output records
        Call SetAllOutputRecords(AssetMonitor_IN)
        ' if any problems were encountered a description is stored in
        ' .LogicBlockParameter9
        .Status = .LogicBlockParameter9
        ' set subconditions, problems will be displayed with .status
        For i = 1 To 13
            If .LogicBlockParameter5(i) = -1 Then
                Set oSubCondition = m_utils.SetCurrentSubCondition(AssetMonitor_IN,
i, 0, False, badLastKnownValue, .Status, False)
            Else
                Set oSubCondition = m_utils.SetCurrentSubCondition(AssetMonitor_IN,
i, .LogicBlockParameter5(i), False, .StatusQuality, .Status, False)
            End If
        Next

        If Err.Number <> 0 Then
            Debug.Assert False
            .Status = "Unhandled runtime error in ILogic_ExecuteLogic():
" & Err.Description
            .StatusQuality = qualityStatusENUM.bad
            Err.Clear
        End If
    End With
    On Error GoTo 0
End Sub


Private Sub ILogic_Terminate()
    Set m_utils = Nothing
End Sub


Private Sub AssetMonitorConfigurationChanged(AssetMonitor_IN As AssetMonitor)
' Perform Configuration Change handling here.
```

```
' This method is used to validate the Asset Monitor configuration such as
' Asset Parameters and Input Records.
' This method is called only if the AssetMonitor_IN.ConfigurationChanged flag is set.
' The ConfigurationChanged flag is set upon the very first execution or if the
' Asset Monitor COnfiguration is modified
' during runtime.

    Dim pos As Integer                  ' position in batch
    Dim H(1 To 13) As Integer           ' hypothesis vector
    Dim indices(1 To 38) As Double      ' index vector
    Dim y() As Double                   ' process variable batch
    Dim r() As Double                   ' set point batch
    Dim u() As Double                   ' control signal batch
    Dim loopCategory As String          ' loop category
    Dim cascade As String               ' slave or master cascade loop
    Dim Ts As Double                    ' sampling time
    Dim OPmin As Double                 ' The minimum control signal value
    Dim OPmax As Double                 ' The maximum control signal value
    Dim LRmin As Double                 ' The minimum output value
    Dim LRmax As Double                 ' The maximum output value
    Dim N As Double                     ' size of the batch
    Dim LCandCascade(1 To 2) As String  ' vector containing loopcategory and
                                        ' cascade values
    Dim TsOPLRandN(1 To 6) As Double    ' vector containing Ts, OPmin/max,
                                        ' LRmin/max and N
    Dim i As Double                     ' used in For loops

    On Error Resume Next

        With AssetMonitor_IN

            For i = 1 To 13
                H(i) = -1               ' value of all hypothesis unknown
            Next
            For i = 1 To 38
                indices(i) = -99999     ' value of all indices unknown
            Next

            ' get asset parameter values
            loopCategory = CStr(.AssetParameters.selectSingleNode
("LoopCategory").Text)
            cascade = CStr(.AssetParameters.selectSingleNode("Cascade").Text)
            N = CStr(.AssetParameters.selectSingleNode("N").Text)
            Ts = CStr(.AssetParameters.selectSingleNode("Ts").Text)
            OPmin = CStr(.AssetParameters.selectSingleNode("OPmin").Text)
            OPmax = CStr(.AssetParameters.selectSingleNode("OPmax").Text)
```

```
            LRmin = CStr(.AssetParameters.selectSingleNode("LRmin").Text)
            LRmax = CStr(.AssetParameters.selectSingleNode("LRmax").Text)

            ' store in vektors, so that fewer .LBP are used
            LCandCascade(1) = loopCategory
            LCandCascade(2) = cascade
            TsOPLRandN(1) = Ts
            TsOPLRandN(2) = OPmin
            TsOPLRandN(3) = OPmax
            TsOPLRandN(4) = LRmin
            TsOPLRandN(5) = LRmax
            TsOPLRandN(6) = N

            ' reset batch vektors
            pos = 1
            ReDim y(1 To N)
            ReDim r(1 To N)
            ReDim u(1 To N)
            For i = 1 To N
                y(i) = -99999
                u(i) = -99999
                r(i) = -99999
            Next


            ' store values in .LBP
            .LogicBlockParameter9Desc = "Errors"     ' variable containging errors
            .LogicBlockParameter9 = ""               ' initially empty

            .LogicBlockParameter1Desc = "Loopcategory and slave or master
cascade loop"
            .LogicBlockParameter1 = LCandCascade
            .LogicBlockParameter2Desc = "Sampling time, OPmin/max, LRmin/max and
batch size"
            .LogicBlockParameter2 = TsOPLRandN
            .LogicBlockParameter3Desc = "Position in the batch"
            .LogicBlockParameter3 = pos
            .LogicBlockParameter4Desc = "Indices calculated for control loop during
last batch"
            .LogicBlockParameter4 = indices
            .LogicBlockParameter5Desc = "Hypotheses calculated for control loop
during last batch"
            .LogicBlockParameter5 = H
            .LogicBlockParameter6Desc = "Vector contaning control signals"
            .LogicBlockParameter6 = u
            .LogicBlockParameter7Desc = "Vector containing process variables"
```

```
            .LogicBlockParameter7 = y
            .LogicBlockParameter8Desc = "Vector containing set points"
            .LogicBlockParameter8 = r

            ' acknowladge that values have been updated
            Call .ConfigurationChangedAck

            If Err.Number <> 0 Then
                Debug.Assert False
                .LogicBlockParameter9 = .LogicBlockParameter9 & "Unhandled runtime
error in AssetMonitorConfigurationChanged(): " & Err.Description
                Err.Clear
            End If

        End With

        On Error GoTo 0
End Sub


Private Sub SetAllOutputRecords(AssetMonitor_IN As AssetMonitor)

    ' variable used to ensure that parameter is String in WriteToOutputRecord
    Dim IndexValueAsString As String

    On Error Resume Next
        With AssetMonitor_IN

                    ' write all outputs
                    IndexValueAsString = .LogicBlockParameter4(1)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "mean_CE", False)
                    IndexValueAsString = .LogicBlockParameter4(2)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "stdev_CE_norm", False)
                    IndexValueAsString = .LogicBlockParameter4(4)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "mean_PV", False)
                    IndexValueAsString = .LogicBlockParameter4(5)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "stdev_OP", False)
                    IndexValueAsString = .LogicBlockParameter4(6)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "skew_CE_norm", False)
                    IndexValueAsString = .LogicBlockParameter4(7)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
```

```
IndexValueAsString, "kurt_CE_norm", False)
                    IndexValueAsString = .LogicBlockParameter4(8)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "maxbicoher_CE", False)
                    IndexValueAsString = .LogicBlockParameter4(11)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "ratio_CE_OP", False)
                    IndexValueAsString = .LogicBlockParameter4(12)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "mode_automatic", False)
                    IndexValueAsString = .LogicBlockParameter4(13)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "mode_saturation", False)
                    IndexValueAsString = .LogicBlockParameter4(14)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "mode_cascade", False)
                    IndexValueAsString = .LogicBlockParameter4(17)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "shutoffvalue_PV", False)
                    IndexValueAsString = .LogicBlockParameter4(18)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "oscillation_CE", False)
                    IndexValueAsString = .LogicBlockParameter4(20)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "osc_period_time", False)
                    IndexValueAsString = .LogicBlockParameter4(21)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "osc_period_freq", False)
                    IndexValueAsString = .LogicBlockParameter4(22)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "osc_amplit_time", False)
                    IndexValueAsString = .LogicBlockParameter4(23)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "osc_amplit_freq", False)
                    IndexValueAsString = .LogicBlockParameter4(24)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "osc_severity", False)
                    IndexValueAsString = .LogicBlockParameter4(25)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "flag_stiction1", False)
                    IndexValueAsString = .LogicBlockParameter4(26)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "flag_stiction2", False)
                    IndexValueAsString = .LogicBlockParameter4(28)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "valve_travel", False)
```

```
                    IndexValueAsString = .LogicBlockParameter4(29)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "ACF_ratio_index", False)
                    IndexValueAsString = .LogicBlockParameter4(32)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "crossing_SP", False)
                    IndexValueAsString = .LogicBlockParameter4(33)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "tracking_SP", False)
                    IndexValueAsString = .LogicBlockParameter4(34)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "travel_ratio_SP_PV", False)
                    IndexValueAsString = .LogicBlockParameter4(35)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "control_performance", False)
                    IndexValueAsString = .LogicBlockParameter4(38)
                    .Status = .Status & m_utils.WriteToOutputRecord(AssetMonitor_IN,
IndexValueAsString, "outlier", False)

                If Err.Number <> 0 Then
                    Debug.Assert False
                    .LogicBlockParameter9 = "Unhandled runtime error in
SetAllOutputRecords(): " & Err.Description
                    Err.Clear
                End If

        End With

        On Error GoTo 0
End Sub
```

Bibliography

[1] ABB. *Optimize^{IT} Loop Performance Manager User Manual*. 2004.

[2] Horch, Alexander. *ConditionMonitoring of Control Loops*. PHD thesis, Department of signals, sensors and systems, Royal Institute of Technology, Stockholm, 2000.

[3] Biao Huang and Sirish L. Shah. *Performance assessment of control loops: theory and applications*, Department of Chemical and Materials Engineering, University of Alberta, Edmonton, Alberta, Canada, T6G 2G6. 1999.

[4] ABB. *Asset Monitor SDK User's Guide*. WE-DOC-04562-C Litho, U.S.A. 2003.

[5] The MathWorks. *MATLAB COM Builder User's Guide*. 2002.

[6] M.A.A.S. Choudhury*, N.F. Thornhill** and S.L. Shah**. *A data-driven model for valve stiction*,*Department of Chemical and Materials Engineering University of Alberta, Edmonton AB, Canada, T6G 2G6, **Department of Electronic and Electrical Engineering University College London, UK, WC1E 7JE