Collecting, storing and reporting production process data in industrial applications.

Author:	Peter Tiagunov
Technical editor and consultant:	John Weigand
Ladder logic design:	John Machnack
Stored procedures design:	Eric Chaves, Peter Tiagunov
Report design:	Peter Tiagunov

The purpose of this document is to provide a relatively detailed view on the theory of collecting and processing production data using available technologies.
This document focuses on the methods of preparing and collecting data using the PLC5 or SLC5 Allen Bradley industrial controllers and software tools provided by Rockwell Software Inc.
A similar approach can be used for other devices and software within a custom application environment.

Theory of operation:

A production process may contain a lot of different parameters or variables that can provide important information about the process to help identify potential problems and help correct them. Examples of such parameters or variables are: temperature, pressure, weight, voltage, current, speed, torque, angle, dimension, or practically any analog or discrete *value* that can be electronically read into an industrial controller. A production process in most cases, is a continuous cycle process where the same activity takes place over and over again. Collection of the history about the process or maybe about multiple processes that take place around the same time is the key to learning and understanding the process(s) and their interrelations.
There are 3 important components that describes the electronic data collection system:

1. The Data originator
2. The Collector
3. The Data Store

Note: *The reporting portion is not a part of the electronic data collection system, but it can be applied at any level where a Data Store is available.*

The ***Data originator*** – is a particular *production operation* (process) itself and data (samples, setpoints) that will be collected for the operation (*). In other words the Data originator can be described as a combination of sensors, transducers, switches, photoeyes etc., with controller input modules (cards) and logic (if any) that process the inputs and produces the data.
The ***Collector*** – is a *mechanism* that will *transform* (optional) and *deliver* the samples from the Data originator to the Data Store.
The ***Data Store*** – is a *place* where the samples are going to be stored, temporarily or for a long period of time. The amount of samples can be defined (buffer) or it can be virtually unlimited (db table).
(*) *The Data Store can be used as a source (link) for another Data originator as well. The links can be reapplied as many times as needed (Fig.1). The format in which data presented in the Data Store may change when the transformation is taking place in the Collector.*

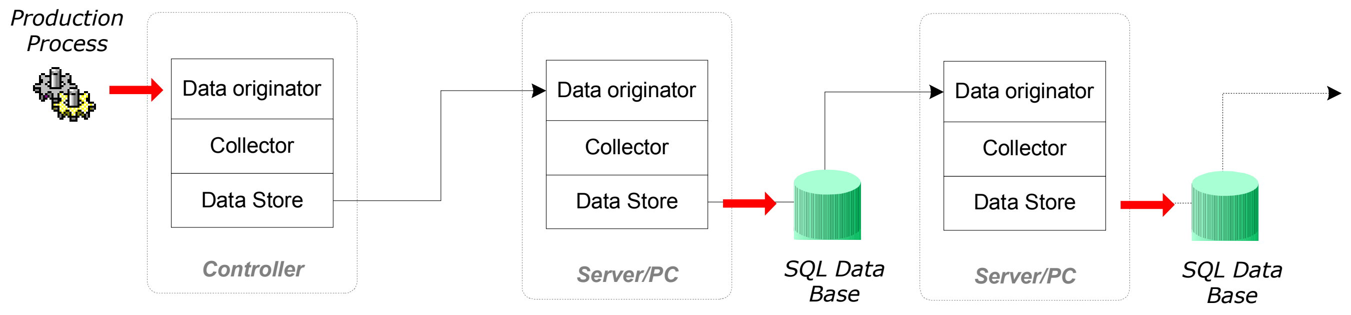


Fig.1

This approach allows a greater flexibility and separation of components resulting in a faster deployment, debugging and maintenance. Using data transformation enables the Collector to focus on a particular aspect of the data.

Samples – a minimum amount of meaningful production data collected for a single subject of production. Example of a sample: *Max: 10, Min: 20, Actual: 15, Time: 01-01-2001 12:59:59.*

Sample(s) could be collected on an event or a time interval basis. This document focuses on event based sampling, when a sample collection is triggered by a specific event (trigger). The time interval sampling or scanning method can be treated as event based sampling where an event is scheduled to trigger periodically.

Every time when a sample is taken, it becomes available in the Data originator. It is the responsibility of the Collector to deliver the sample to the Data Store successfully. When data collection is critical it should be treated as part of the production process. When sample(s) cannot be delivered successfully to the Data Store, the production process must be interrupted until the sample(s) will reach their destination successfully.

The destination in this case could be the first Data Store or Data Store located at the next level or levels. Confirmation must be returned to production process when the sample(s) successfully reached the Data Store at the desired level.

When the production process has a high-speed cycle time <5 sec. the Data Store must accumulate (buffer) some samples prior to passing them to the next level (Fig.2). While samples are passing to the higher level the Collector can continue to add new samples to the buffer (Data Store). Sending data in buffered chunks minimizes communication overhead. To avoid duplication, the Collector in the higher level checks the samples before adding them to the Data Store at that level.

When the production process has a slow-speed cycle time >5 sec., it is possible to allow the Collector to acquire a *single current* sample remotely over the network (Fig.3). This approach is not applicable in public networks and should only be used if a designated connection or networks were QoS (quality of service) was provided.

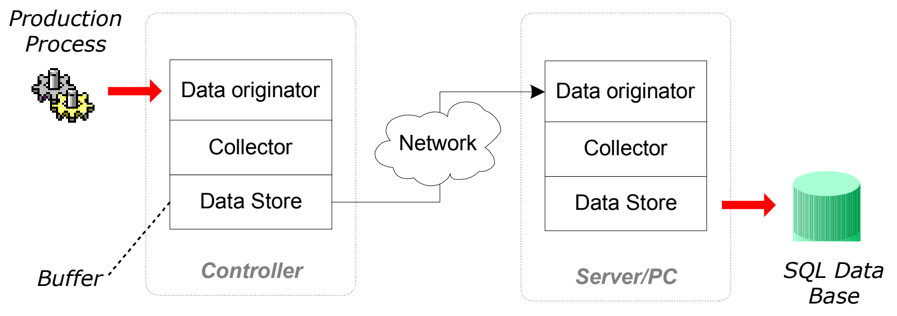


Fig.2

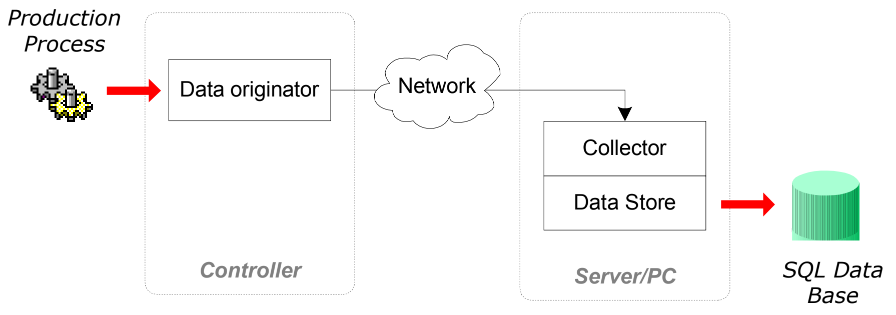


Fig.3

Case Study:

Requirements: Collect, store and report induction heater temperature samples for each part. Data collection is not critical.

Controllers used: SLC5/04 or PLC5 Allen Bradley.
Sensor: IRCON SR series.
Cycle Time: 5-6 sec.
Peak temperature samples received in the PLC via the analog input module.

Approach:
Temperature samples from the analog module (Data originator) will be stored in the SLC memory's buffer (Data Store) using the program written in ladder code (Collector). The buffer will hold up to 32 samples (Fig.4). Each sample includes the following data:

1. Index (incrementing number 1-32767)
2. Lo temperature set point
3. Hi temperature set point
4. Peak temperature read
5. Date and time when sample was taken
6. (Optional - Part Number information)

SLC File	N15	N15	N15	N15	N16	N16	N16	N16	N16	N16
Row	*Index*	*Lo*	*Peak*	*Hi*	*MM*	*DD*	*YYYY*	*Hrs*	*Min*	*Sec*
0	N15:0	N15:32	N15:64	N15:96	N16:0	N16:32	N16:64	N16:96	N16:128	N16:160
1	N15:1	N15:33	N15:65	N15:97	N16:1	N16:33	N16:65	N16:97	N16:129	N16:161
29	N15:29	N15:61	N15:93	N15:125	N16:29	N16:61	N16:93	N16:125	N16:157	N16:189
30	N15:30	N15:62	N15:94	N15:126	N16:30	N16:62	N16:94	N16:126	N16:158	N16:190
31	N15:31	N15:63	N15:95	N15:127	N16:31	N16:63	N16:95	N16:127	N16:159	N16:191

Fig.4

The collector keeps writing new samples to the buffer. When all 32 rows are filled the collector will continue to write to the buffer starting from row with address 0. Every time the collector updates the row with address 31 it will start again with row address 0 for the following sample. During this process when the fill threshold is reached the collector will raise the flag – indicating that new samples are collected and ready for processing (**Fig.5**). The fill threshold is a calculated variable that allows raising the flag when buffer is partially filled; for example at 85%, when the flag is raised, the next threshold will be calculated for the next 85% beginning with the row where the last one was identified.
This method ensures that there are always a few (5-6) records kept from the previous collection cycle. Having duplicate samples sent to the next level guarantees that there will be no missing samples. Samples could be missed while buffer is being processed to the next level or samples can be overwritten with new ones if buffer processing is taking more time than the actual production cycle. For a production cycle of 5 sec., 85% threshold is optimum. It allows keeping about 5 records overlap – equivalent to 25 seconds. This safe zone provides sufficient time to retry buffer processing (Update Cycle) a few times in case if first time process failed. Failure can occur as a result of temporary network outage or some other errors. The faster the cycle time the smaller the fill threshold should be. Typically a 25 sec. safety zone is sufficient. Example of the fill threshold calculation shown in (Fig.6).

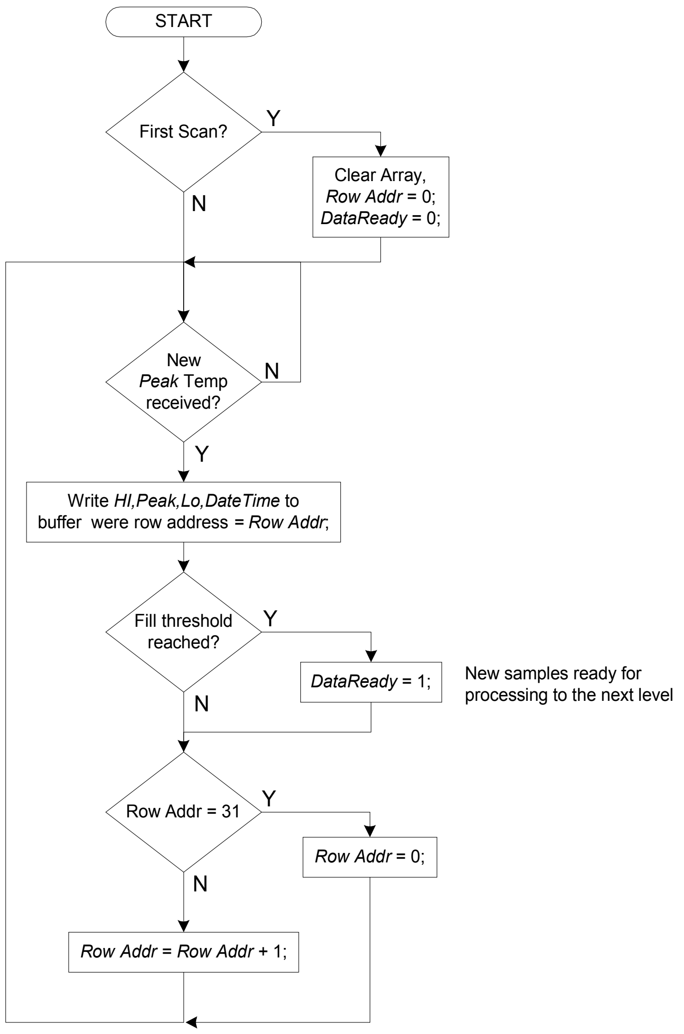

START
First Scan?
Y
N
Clear Array, Row Addr = 0; DataReady = 0;
New Peak Temp received?
N
Y
Write HI,Peak,Lo,DateTime to buffer were row address = Row Addr;
Fill threshold reached?
Y
N
DataReady = 1;
New samples ready for processing to the next level
Row Addr = 31
Y
N
Row Addr = 0;
Row Addr = Row Addr + 1;

Fig.5

Avg. Cycle time, sec.	Safety Zone, sec.	Overlap samples	Calculated Threshold, % (Buffer size = 8)	Calculated Threshold, % (Buffer size = 16)	Calculated Threshold, % (Buffer size = 32)	Calculated Threshold, % (Buffer size = 64)
1.0	25.0	25	*Increase Buffer*	*Increase Buffer*	21.88	60.94
1.5	25.0	16	*Increase Buffer*	*Increase Buffer*	47.92	73.96
2.0	25.0	12	*Increase Buffer*	21.88	60.94	80.47
2.5	25.0	10	*Increase Buffer*	37.50	68.75	84.38
3.0	25.0	8	*Increase Buffer*	47.92	73.96	86.98
3.5	25.0	7	10.71	55.36	77.68	88.84
4.0	25.0	6	21.88	60.94	80.47	90.23
4.5	25.0	5	30.56	65.28	82.64	91.32
5.0	25.0	5	37.50	68.75	84.38	92.19
5.5	25.0	4	43.18	71.59	85.80	92.90
6.0	25.0	4	47.92	73.96	86.98	93.49
6.5	25.0	3	51.92	75.96	87.98	93.99
7.0	25.0	3	55.36	77.68	88.84	94.42
7.5	25.0	3	58.33	79.17	89.58	94.79
8.0	25.0	3	60.94	80.47	90.23	95.12
8.5	25.0	2	63.24	81.62	90.81	95.40
9.0	25.0	2	65.28	82.64	91.32	95.66
9.5	25.0	2	67.11	83.55	91.78	95.89
10.0	25.0	2	68.75	84.38	92.19	96.09

Fig.6 *Example of fill threshold calculation table*

The following example (Fig.7) illustrates how the threshold logic will perform under the following conditions:
Production Avg. Cycle: *9.0 sec*.
Safety Zone: *25 sec.*
Buffer size: *8 samples (see Fig.6, marked in red).*

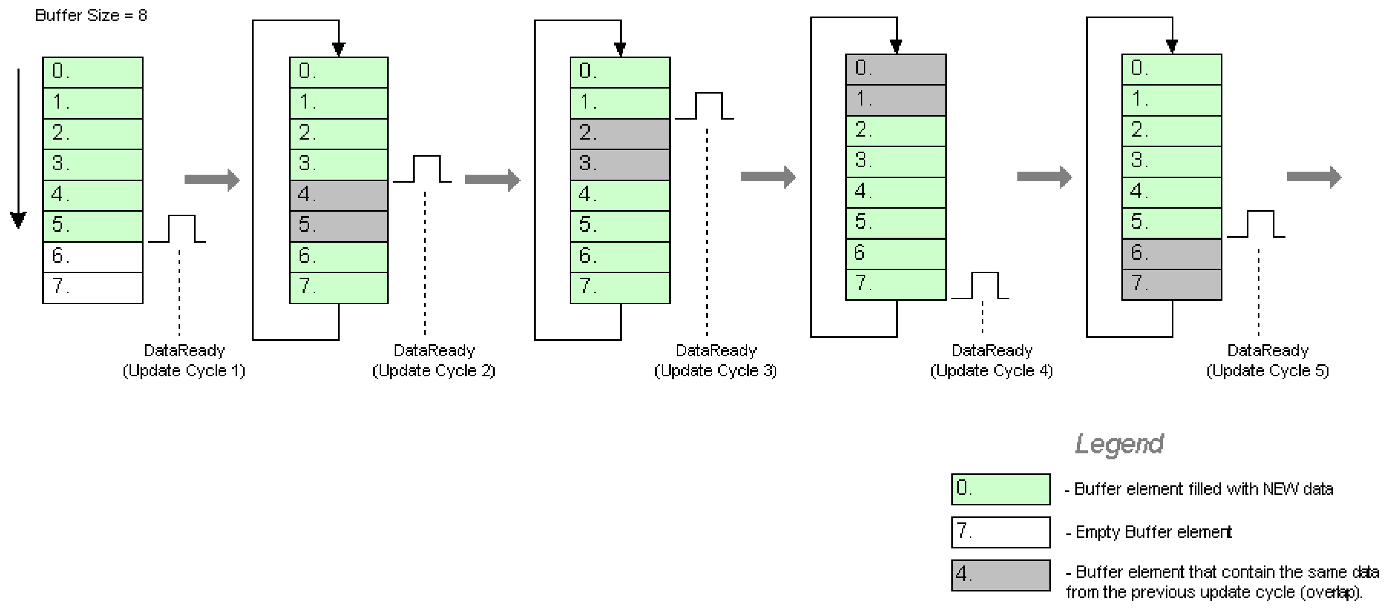

Buffer Size = 8
0.
1.
2.
3.
4.
5.
6.
7.
DataReady (Update Cycle 1)
DataReady (Update Cycle 2)
DataReady (Update Cycle 3)
DataReady (Update Cycle 4)
DataReady (Update Cycle 5)
Legend
- Buffer element filled with NEW data
- Empty Buffer element
- Buffer element that contain the same data from the previous update cycle (overlap).

Fig.7 *Overview of fill threshold logic cycles*

Note: *In applications where data collection is critical, the production process must be stopped when safety zone (overlap) time has elapsed and a confirmation about a successful update cycle has not been received. This will guarantee that the non-sent samples in the buffer will not be overwritten.*

For more details on the controller level collector and threshold see SLC ladder code in **Attachment A.**

At this point the portion of collecting samples at the controller level is covered. The next part will explain what happenes when he "*DataReady"* flag is raised and how the Update Cycle will process the samples to the next level. The following illustration provides an overview of the core elements of an electronic data collection system (Fig.8)

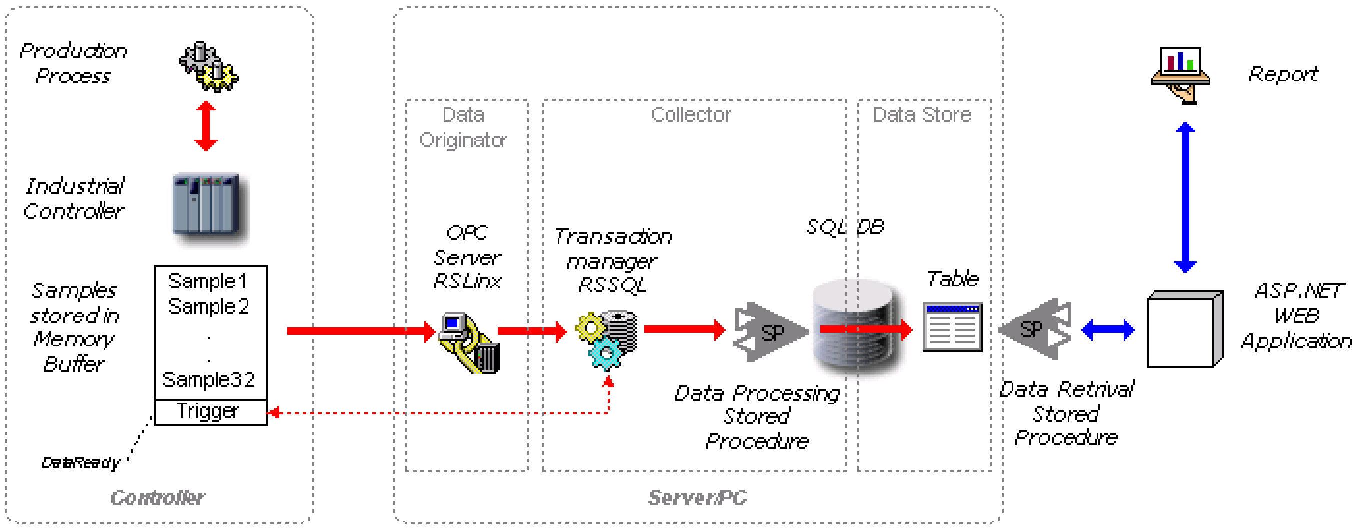
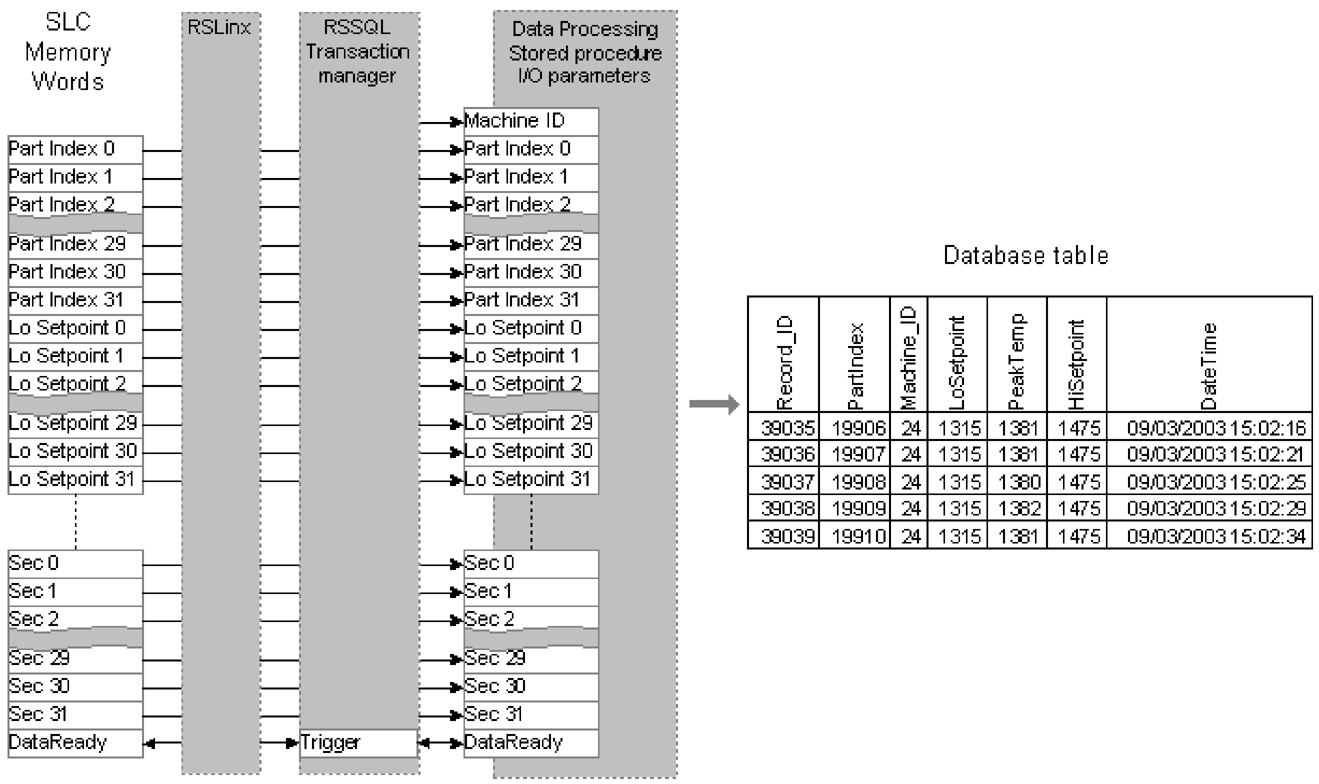


Fig.8 *The core elements of an electronic data collection system*

The Transaction Manager is the key component of the system. In conjunction with the Data Processing Stored Procedure they represents the Collector. In this example, the RSSQL product (Rockwell Software Inc.) will be used as the transaction manager. http://www.software.rockwell.com/rssql/.
Note: *RSSQL includes generic OPC connector that can be used with third-party OPC servers.*
The RSSQL will use the SQL stored procedure to write samples to the data base. "A stored procedure is a group of Transact-SQL statements compiled into a single execution plan." –Microsoft SQL Server Books Online. The stored procedure used in this example will utilize input and output parameters. I/O parameters provide flexible external interface for communication with another application. The RSSQL will connect the samples data in the SLC to the Data Processing Stored procedure where the data would be transformed and then stored in the database table. An overview of this interface shown in (Fig.9).



Record_ID	PartIndex	Machine_ID	LoSetpoint	PeakTemp	HiSetpoint	DateTime
39035	19906	24	1315	1381	1475	09/03/2003 15:02:16
39036	19907	24	1315	1381	1475	09/03/2003 15:02:21
39037	19908	24	1315	1380	1475	09/03/2003 15:02:25
39038	19909	24	1315	1382	1475	09/03/2003 15:02:29
39039	19910	24	1315	1381	1475	09/03/2003 15:02:34

Fig.9 *Interfaces and table design*

The Transaction will be triggered by the HI transition in the "DataReady" word of the SLC(0 to 1). RSSQL will then collects the entire buffer from the SLC, and trigger execution of a stored procedure with input parameters read from the SLC.

Upon successful execution, the stored procedure will reset the trigger by returning 0 to "DataReady" through the output parameter. If the transaction fails or times out Transaction Manager will retry the execution after a predefined period of time, it will continue to retry execution until it is successful. See Data Processing stored procedure code example in **Attachment B**.

Collecting production process data for multi-operational production processes

From a single operation process (heating parts) the focus will be switched to a multi-operational production process. The following describes the multi-operational production process:
A production process where multiple operations with single or multiple subjects of production (parts) performed during the same period of time on identified equipment and/or by identified individual(s) is called multi-operational production process. Two examples of multi-operational production processes are conveyors and assembly lines.

What is critical to the collection process of this type is part tracking. A unique identifier must be assigned to every part that the data is collected for. In the case study, described earlier in this document, the Part Index was used to identify the part. The index number in the case study was assigned temporary and by the time when part left the line any relations between the actual part and index that had been assigned to this part were destroyed. It would be impossible to track back to this particular part. This problem will be solved if each part receives a permanent unique identifier that will remain with the part for its entire life cycle, example: stamp, barcode, engravings, etc. This document focuses on types of processes where unique identifiers were assigned temporarily. If multiple production equipment (machines) were involved in the process, one of the controllers must provide part tracking functionality by maintaining part indexes (identifiers) with their location through the entire production process (Fig.10).

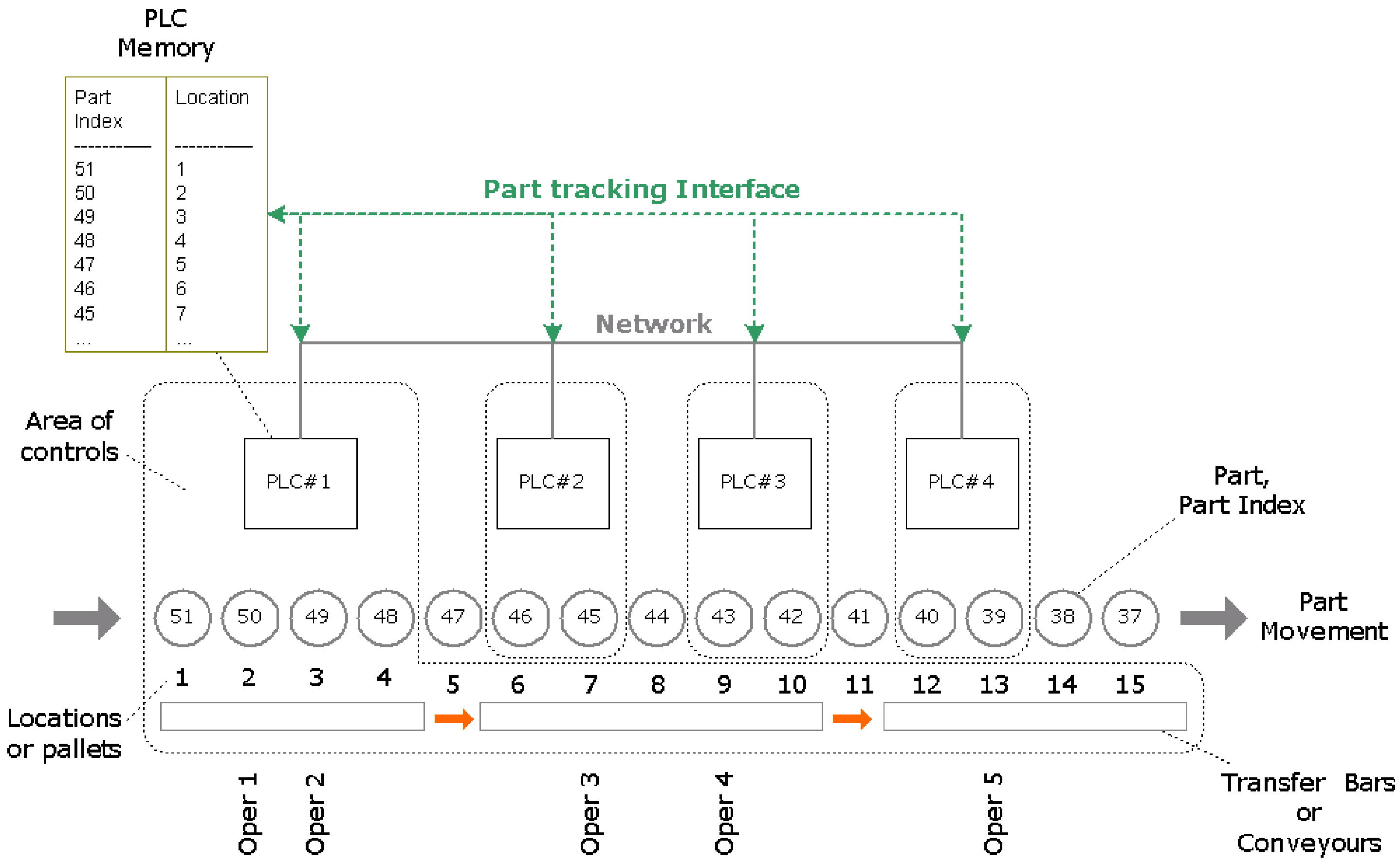


Fig.10

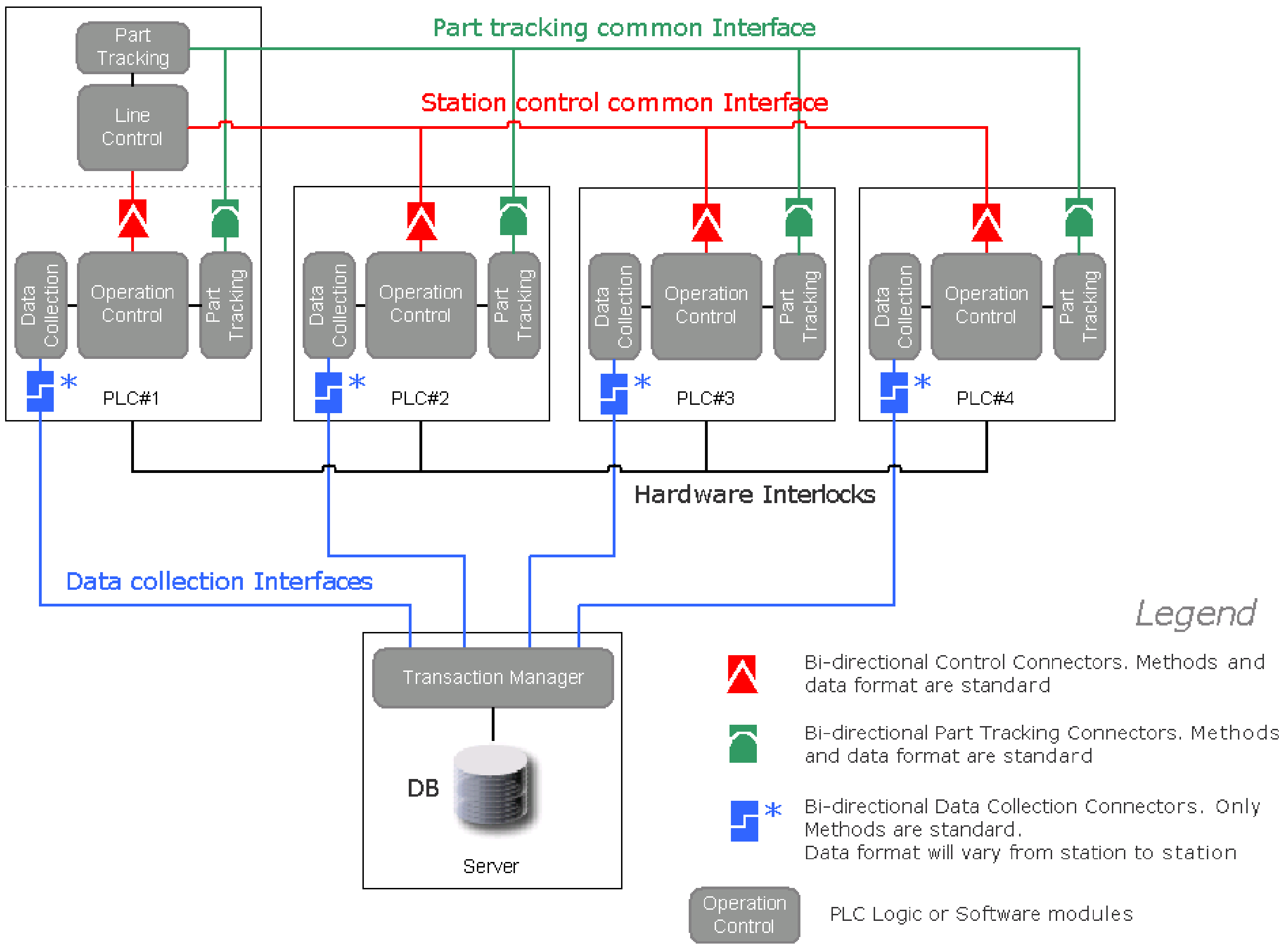
From Fig.10, it is possible to see that the PLC#1 controller is responsible for tracking parts through the entire line. Other PLC's will request and/or update the Part Index number for each part when the operation is ready to be performed by a particular PLC. The important part of the part tracking process is to identify the methods and data format that are going to be used for data exchange between each individual station and central location that hosts the part index and location information. Methods will include answers to the following question examples:

- How the information about part index and its location will be requested and how is it going to be updated?
- What PLC is responsible for initiation request, update?
- How will Part Index be handled for rejected parts?
- What happens if the part(s) get manually removed from the line?
- How will the Part Index be handled if transfer allowed while part is not present at load or unload point?
- How to handle Part Tracking while individual station is running separately from the entire line?

Together, the methods and data format will describe the module and the connector of a Part Tracking Interface. Once developed it can be simply reused through the entire line. The architecture of a connector and module will depend on a particular line design and functional requirements, which will be different for different lines.

Besides the Part Tracking connector the same approach can be used in a development of a control connector that allows an exchange of standard sets of commands and states between the line control modules and each individual station. Once again the control connector logic can be reused through the entire line (Fig.11).

Note: *The same interfaces are used between the modules located on different PLC's and the modules located on the same PLC (PLC#1). Physical networks, such as Ethernet, serve as a medium to connect the modules between remote PLC's.*



Part tracking common Interface
Station control common Interface
Part Tracking
Line Control
Data Collection
Operation Control
Part Tracking
PLC#1
PLC#2
PLC#3
PLC#4
Hardware Interlocks
Data collection Interfaces
Transaction Manager
DB
Server
Legend
Bi-directional Control Connectors. Methods and data format are standard
Bi-directional Part Tracking Connectors. Methods and data format are standard
Bi-directional Data Collection Connectors. Only Methods are standard. Data format will vary from station to station
Operation Control
PLC Logic or Software modules

Fig.11

The Data Collection Connector, in the example described above, in comparison to the Control Connector and the Part Tracking connector, has only the methods that are standard, while the format of collected data will differ from station to station. Differences in data format are a result from different types of operations and different types of samples that can be collected; for example tonnage, speed, torque and limits associated with these parameters. An example of a Data Collection Connector data format is depicted in (Fig.4). Examples of data collection and data exchange methods are depicted in (Fig.5).

Lets take a look at what is happening when samples are collected and ready to be sent to the data store.

Figure 12 is an overview of a temperature samples collection system. Multiple machines (stations) collecting and buffering samples. Each machine (station) has its own transaction configured in Transaction manager. Once data is ready, the Data originator will trigger the transaction allowing the data to be sent to the database via the stored procedure. In this system, the format of data is consistent across the machines where data is collected. See (Fig.4). In other words the Data Collection connector has standard methods and standard data formats.
Having a standard Data Collection connector for multiple machines allows for the introduction of a single Data Processing stored procedure that can be reused for these machines.

Note: *In the case where data format differences are not significant from machine to machine there is still a possibility to handle these machines via the same stored procedure. Obviously in this case the procedure will become more complex. In order* ***not to overcomplicate*** *the system if the data format differences are significant from one machine to another or from one group of machines to another, it is* ***always*** *better to create two or more separate procedures that specialize in a particular area.*

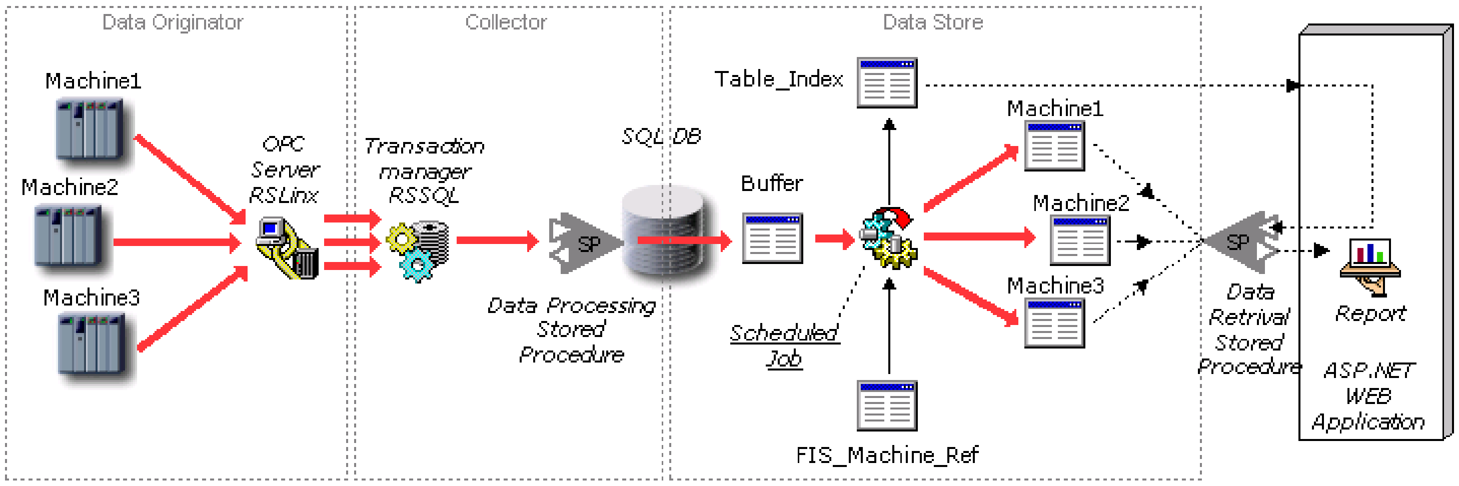


Fig.12 Overview of a Temperature Samples Collection system.

Lets continue the overview of the temperature samples collection system. When a transaction is triggered, the data from the machine will be sent to the buffer table via the stored procedure. The buffer table is designed to temporarily hold records containing a set of the last samples from different machines. Having the last updates (samples) stored temporarily in the buffer prevents duplicate entries (the buffer in the PLC always contain a few overlapping records). Example of this table is shown in (Fig.9 on the left). Another process (Scheduled Job) is designed to sort the records based on theirs Machine_ID numbers and move them to a designated history tables (Fig.12). The same sorting process will create a new table automatically, using the name provided in the FIS_Machine_Ref table, when a new machine is added to the collection system. (See **Attachment C** for code example).

What would be the difference between the temperature samples collection system and data collection system where data format is different from station to station? The data collection system is focused on building process parameters history from a part perspective. In other words it provides a birth history for each part. Compared to temperature samples where the collection system focus is placed on the actual process itself.
Overview of a generic data collection system represented in (Fig.13).

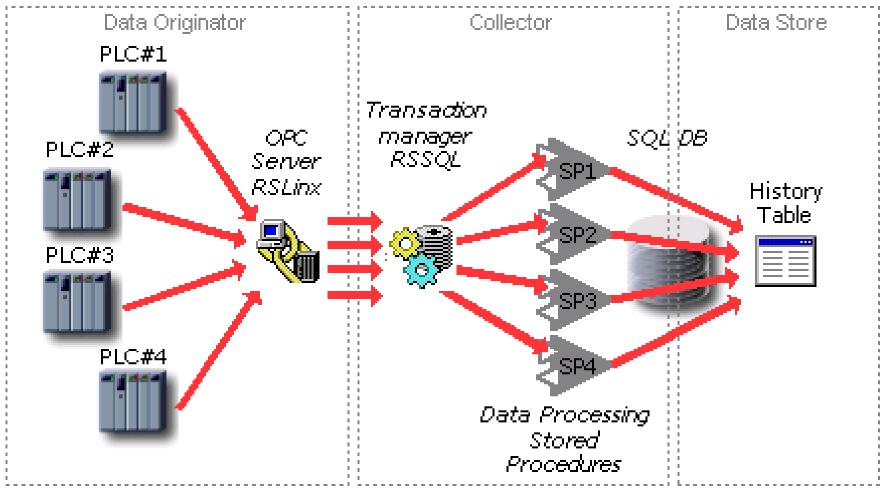


Fig.13 Overview of a generic data collection system

As a result of the data format differences four stored procedures are required, one for each station or operation. The interesting part about the system is that collection processes are completely unparallel and asynchronous. Data for the same part will be sent through four different buffers, transactions and stored procedures. In the history table collected data will reunite in a single table record identified by the Part Index identifier. As mentioned earlier, it is *always* a good practice, but not the goal, to look for opportunity of reusing elements of the system. For example in the case of a generic data collection system, it may allow a reduced number of data processing stored procedures.

The reporting part of the system is an ASP.NET web application with code behind approach. The application is deployed on the IIS server is available via a browser. The same stored procedure is used to retrieve the data. The report application interface allows a selection of different machines, a number of samples to display and a scroll through the history of samples. Example of the temperature collection report screens shown in **Attachment E**.

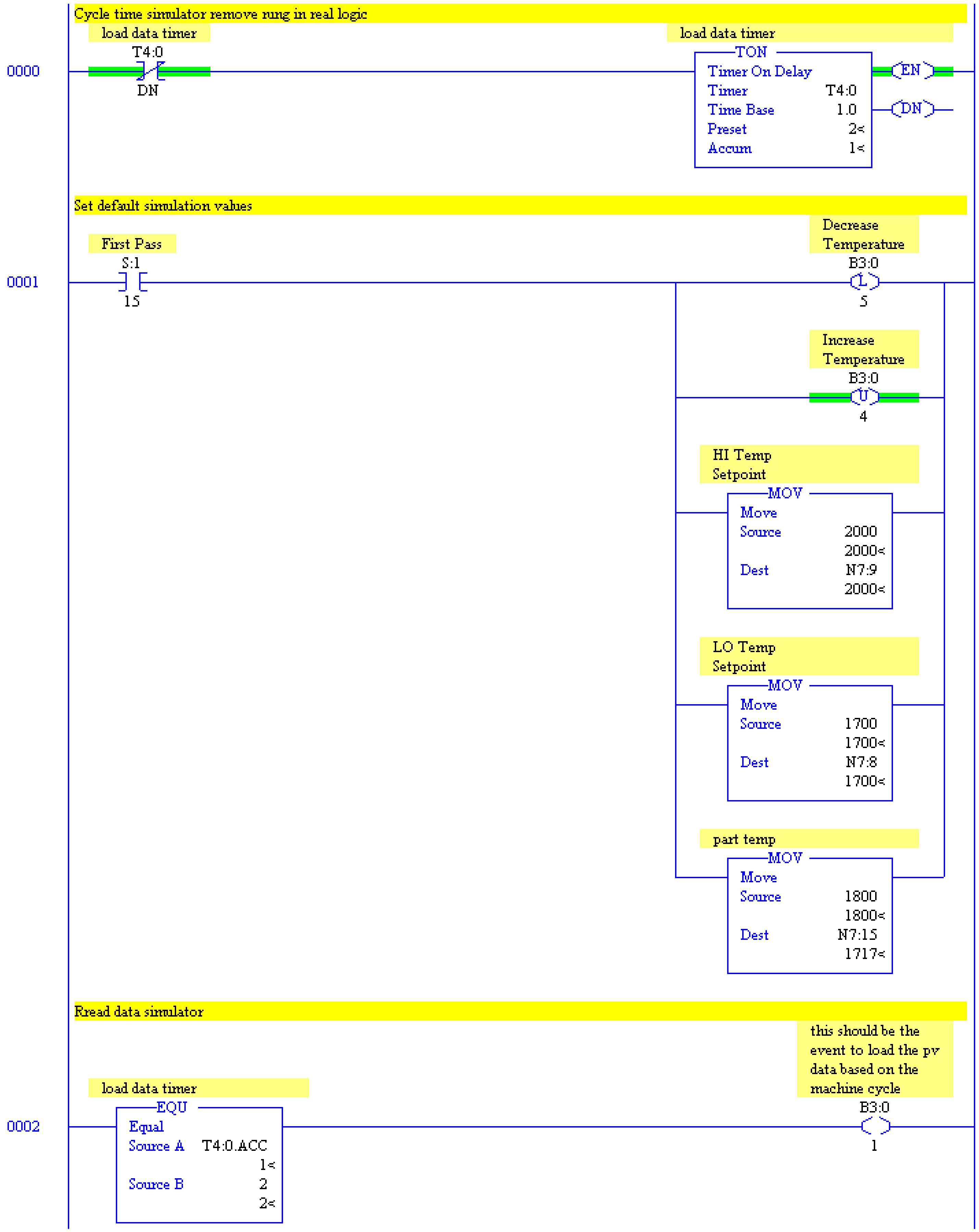
Conclusions

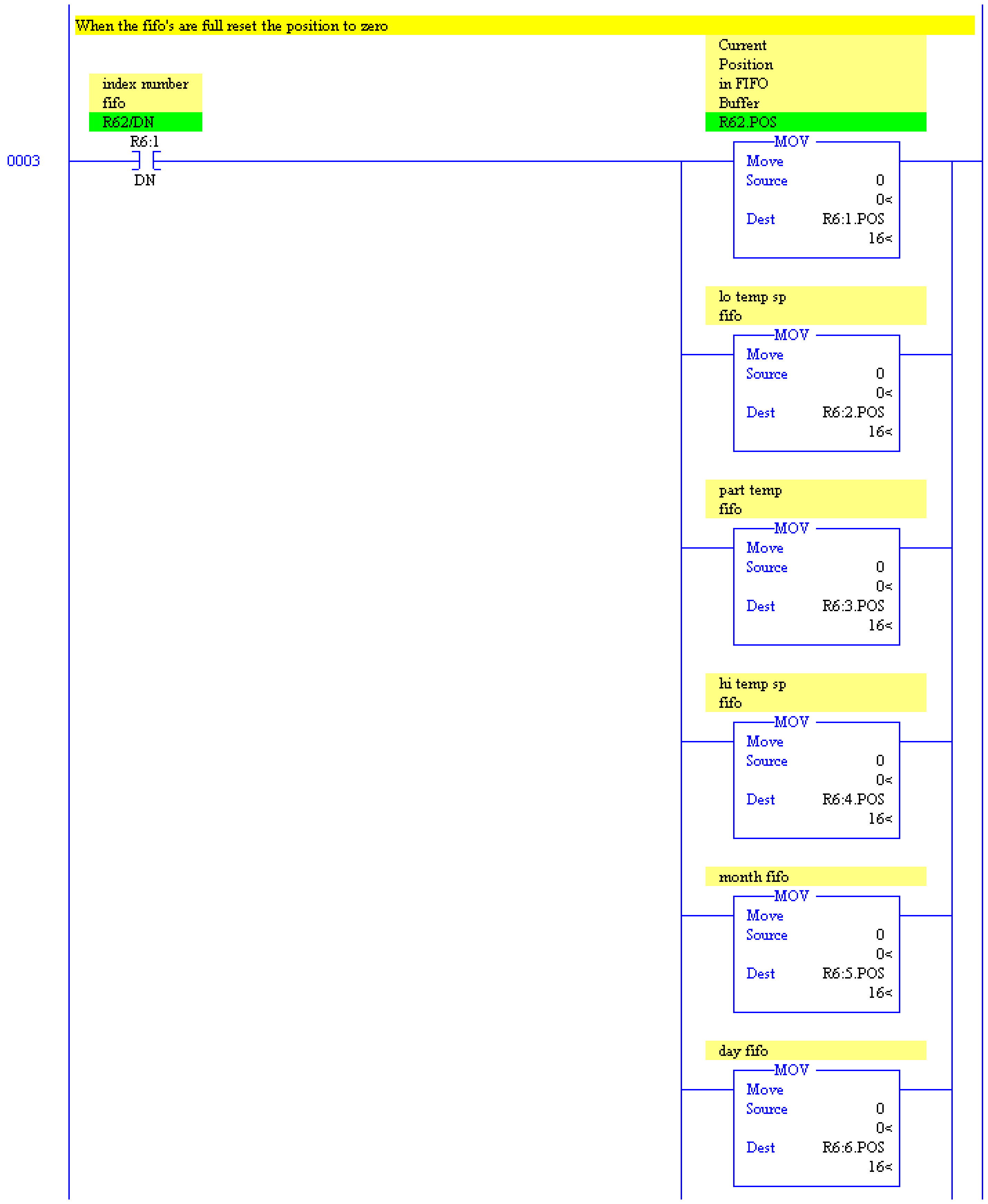
It is a quite common problem (tendency) in the development of a data collection system to design the entire system and its elements from scratch. This should be avoided. Using pre build components or shelf products will reduce the debug time significantly; remember someone already went through the design, stabilization and release steps with it. In two different systems the same components could be connected together in a different way. Improvisation, architecture flexibility, balance between reusability and complexity are the key elements to building a reliable system.

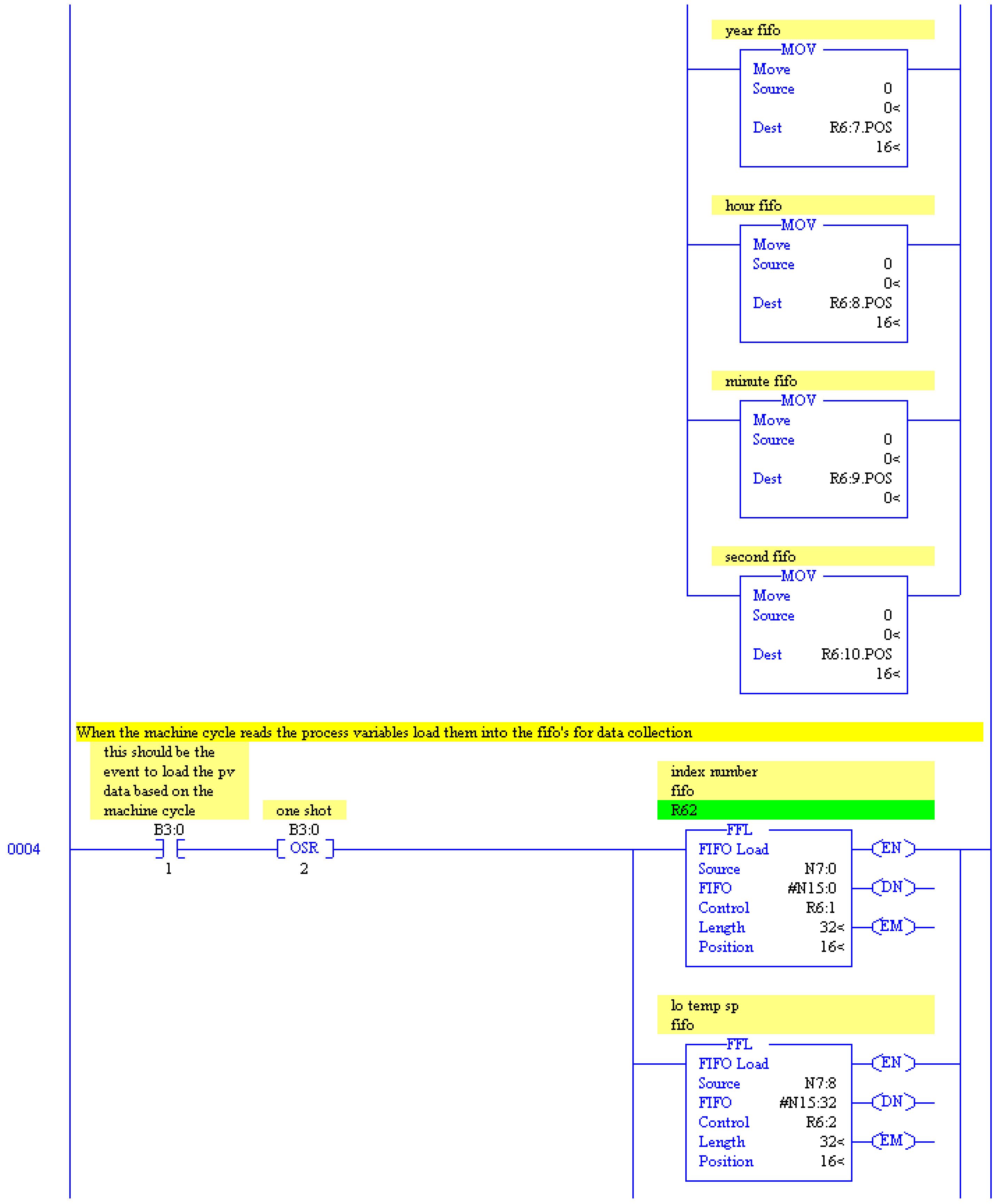
Software, operating system, tools and components involved:

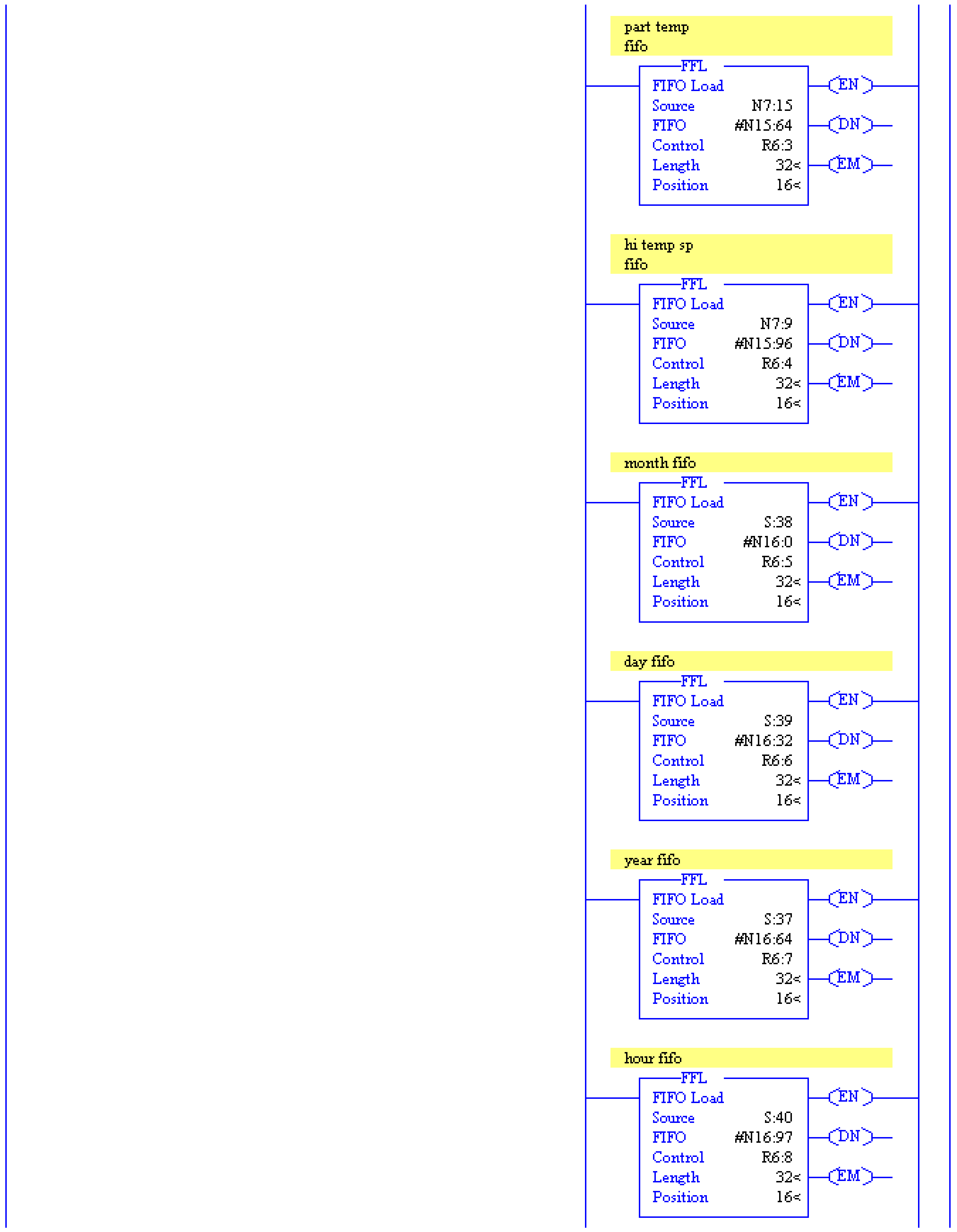
PLC communication software: RSLinx Gateway (Rockwell Software Inc).
PLC Programming software: RSLogix500, RSLogix5 (Rockwell Software Inc).
Transaction Manager: RSSQL (Rockwell Software Inc). http://www.ab.com/
Database: SQL Server 2000 (Microsoft Corporation).
Operating system: Windows 2000 Server (Microsoft Corporation). http://www.microsoft.com/
Chart Generator: PopChart Standard (Corda Technologies Inc). http://www.corda.com/
Web development tool: Visual Studio.NET 2002 (Microsoft Corporation). http://msdn.microsoft.com/vstudio/productinfo/

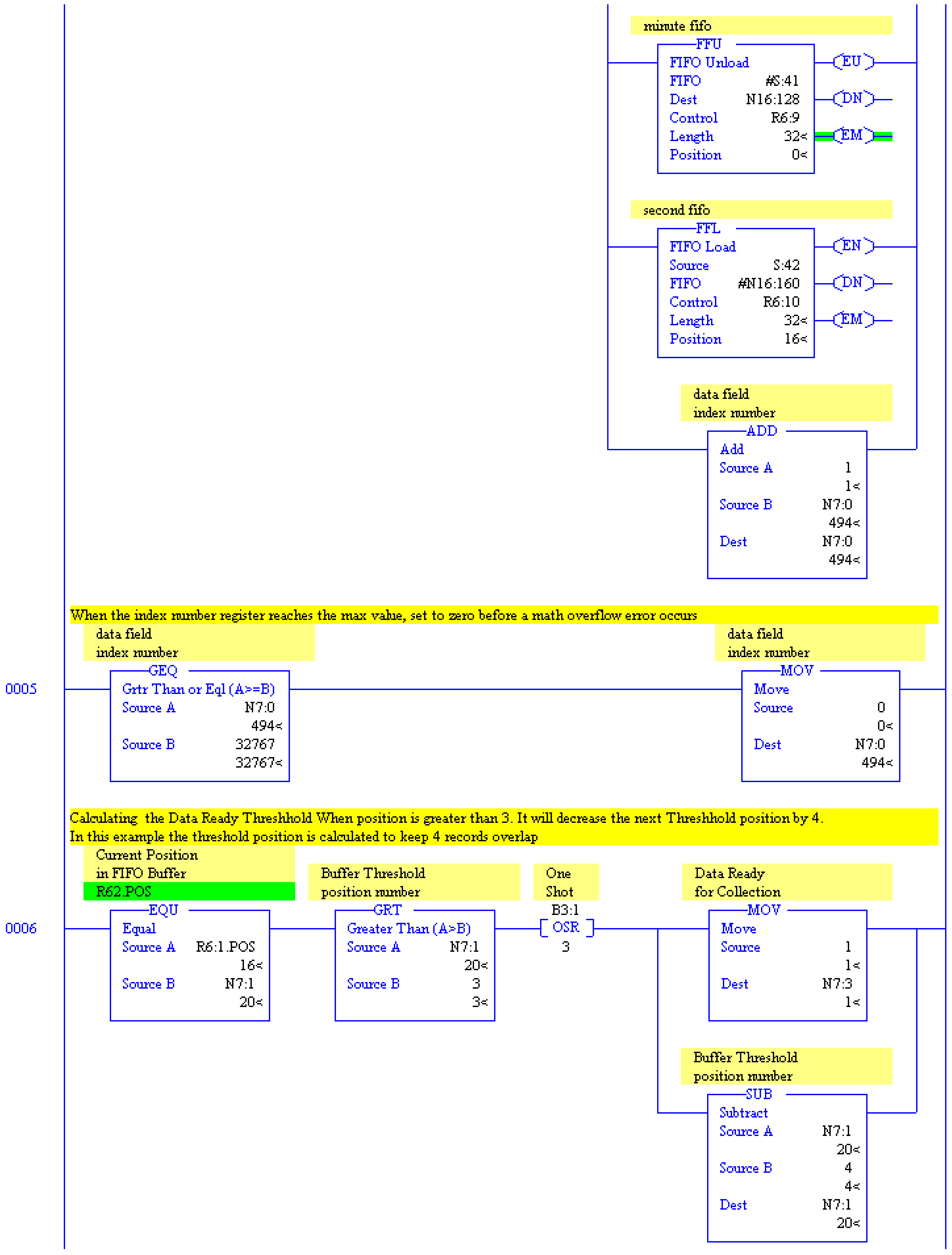
Attachment A. ***Data Collection Simulator. Ladder logic for SLC5 processor example:***




When the fifo's are full reset the position to zero
index number
fifo
R62/DN
R6:1
0003
DN
Current
Position
in FIFO
Buffer
R62.POS
MOV
Move
Source 0
0<
Dest R6:1.POS
16<
lo temp sp
fifo
MOV
Move
Source 0
0<
Dest R6:2.POS
16<
part temp
fifo
MOV
Move
Source 0
0<
Dest R6:3.POS
16<
hi temp sp
fifo
MOV
Move
Source 0
0<
Dest R6:4.POS
16<
month fifo
MOV
Move
Source 0
0<
Dest R6:5.POS
16<
day fifo
MOV
Move
Source 0
0<
Dest R6:6.POS
16<


year fifo
MOV
Move
Source 0
0<
Dest R6:7.POS
16<
hour fifo
MOV
Move
Source 0
0<
Dest R6:8.POS
16<
minute fifo
MOV
Move
Source 0
0<
Dest R6:9.POS
0<
second fifo
MOV
Move
Source 0
0<
Dest R6:10.POS
16<
When the machine cycle reads the process variables load them into the fifo's for data collection
this should be the event to load the pv data based on the machine cycle
one shot
index number fifo
R62
0004
B3:0
1
B3:0
OSR
2
FFL
FIFO Load
Source N7:0
FIFO #N15:0
Control R6:1
Length 32<
Position 16<
EN
DN
EM
lo temp sp fifo
FFL
FIFO Load
Source N7:8
FIFO #N15:32
Control R6:2
Length 32<
Position 16<
EN
DN
EM


part temp
fifo
FFL
FIFO Load
Source N7:15
FIFO #N15:64
Control R6:3
Length 32<
Position 16<
EN
DN
EM
hi temp sp
fifo
FFL
FIFO Load
Source N7:9
FIFO #N15:96
Control R6:4
Length 32<
Position 16<
EN
DN
EM
month fifo
FFL
FIFO Load
Source S:38
FIFO #N16:0
Control R6:5
Length 32<
Position 16<
EN
DN
EM
day fifo
FFL
FIFO Load
Source S:39
FIFO #N16:32
Control R6:6
Length 32<
Position 16<
EN
DN
EM
year fifo
FFL
FIFO Load
Source S:37
FIFO #N16:64
Control R6:7
Length 32<
Position 16<
EN
DN
EM
hour fifo
FFL
FIFO Load
Source S:40
FIFO #N16:97
Control R6:8
Length 32<
Position 16<
EN
DN
EM



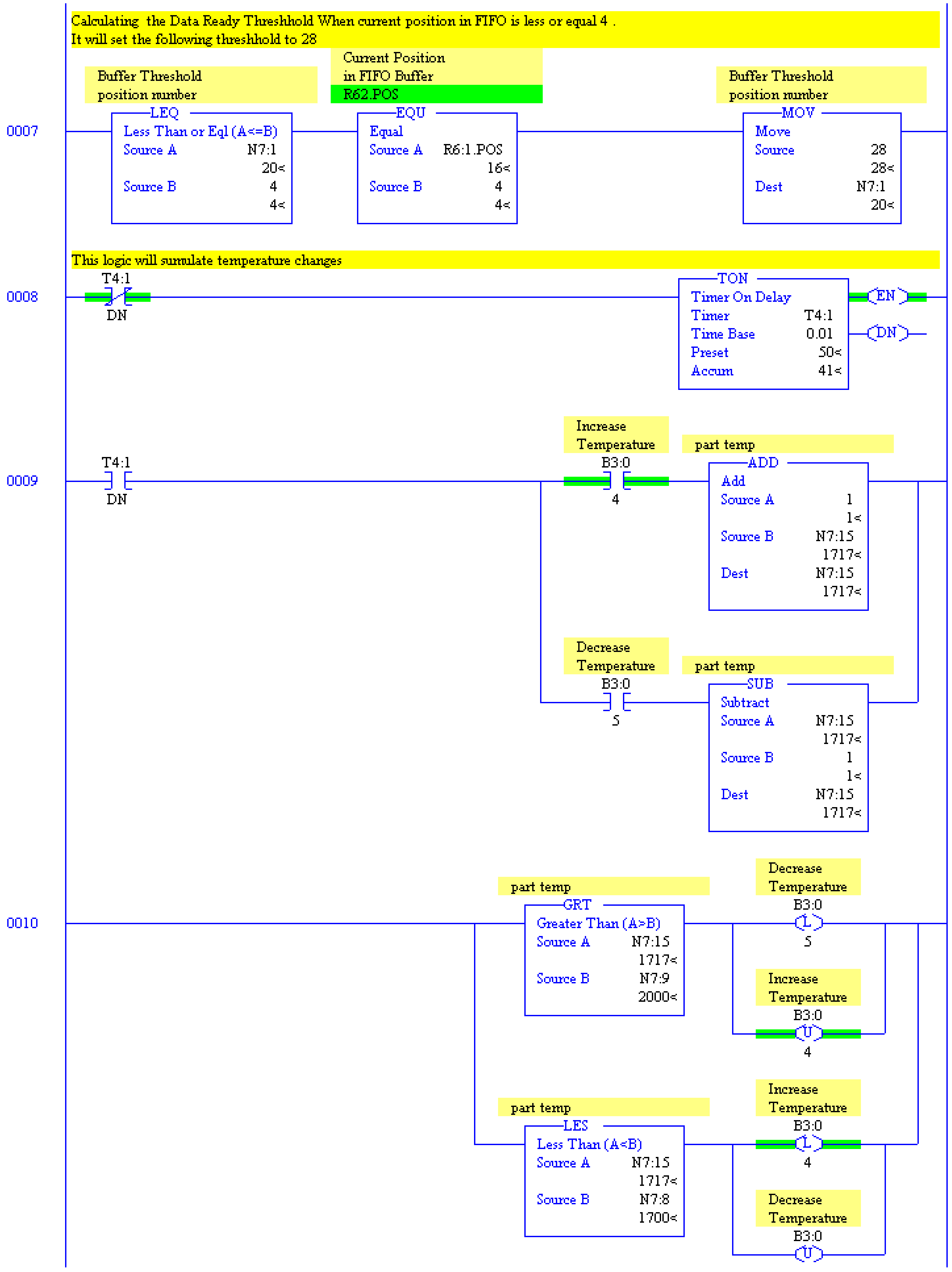
```
minute fifo
FFU
FIFO Unload
FIFO          #S:41        (EU)
Dest          N16:128      (DN)
Control       R6:9         (EM)
Length        32<
Position      0<

second fifo
FFL
FIFO Load                  (EN)
Source        S:42
FIFO          #N16:160     (DN)
Control       R6:10
Length        32<          (EM)
Position      16<

data field
index number
ADD
Add
Source A      1
              1<
Source B      N7:0
              494<
Dest          N7:0
              494<
```

When the index number register reaches the max value, set to zero before a math overflow error occurs

```
0005
data field                              data field
index number                            index number
GEQ                                     MOV
Grtr Than or Eql (A>=B)                 Move
Source A      N7:0                      Source     0
              494<                                 0<
Source B      32767                     Dest       N7:0
              32767<                               494<
```

Calculating the Data Ready Threshhold When position is greater than 3. It will decrease the next Threshhold position by 4.
In this example the threshold position is calculated to keep 4 records overlap

```
0006
Current Position
in FIFO Buffer
R62.POS              Buffer Threshold        One       Data Ready
                     position number         Shot      for Collection
EQU                  GRT                     B3:1      MOV
Equal                Greater Than (A>B)      [OSR]     Move
Source A  R6:1.POS   Source A    N7:1        3         Source    1
          16<                    20<                             1<
Source B  N7:1       Source B    3                     Dest      N7:3
          20<                    3<                              1<

                                                       Buffer Threshold
                                                       position number
                                                       SUB
                                                       Subtract
                                                       Source A  N7:1
                                                                 20<
                                                       Source B  4
                                                                 4<
                                                       Dest      N7:1
                                                                 20<
```



Calculating the Data Ready Threshhold When current position in FIFO is less or equal 4 .
It will set the following threshhold to 28
0007
Buffer Threshold position number
LEQ
Less Than or Eql (A<=B)
Source A N7:1
20<
Source B 4
4<
Current Position in FIFO Buffer
R62.POS
EQU
Equal
Source A R6:1.POS
16<
Source B 4
4<
Buffer Threshold position number
MOV
Move
Source 28
28<
Dest N7:1
20<
This logic will sumulate temperature changes
0008
T4:1
DN
TON
Timer On Delay
Timer T4:1
Time Base 0.01
Preset 50<
Accum 41<
EN
DN
0009
T4:1
DN
Increase Temperature
B3:0
4
part temp
ADD
Add
Source A 1
1<
Source B N7:15
1717<
Dest N7:15
1717<
Decrease Temperature
B3:0
5
part temp
SUB
Subtract
Source A N7:15
1717<
Source B 1
1<
Dest N7:15
1717<
0010
part temp
GRT
Greater Than (A>B)
Source A N7:15
1717<
Source B N7:9
2000<
Decrease Temperature
B3:0
L
5
Increase Temperature
B3:0
U
4
part temp
LES
Less Than (A<B)
Source A N7:15
1717<
Source B N7:8
1700<
Increase Temperature
B3:0
L
4
Decrease Temperature
B3:0
U

Attachment B. ***Data processing T-SQL stored procedure. Code example:***

```
CREATE PROCEDURE sp_Process_TempTracking_data_from_Device
(
@p_Machine_ID  int,
-------------------------------------------
@p_PartIndex00 int,
@p_PartIndex01 int,
.
.
@p_PartIndex30 int,
@p_PartIndex31 int,
--------------------------------------------
@p_LoSetpoint00       int,
@p_LoSetpoint01       int,
.
.
@p_LoSetpoint30       int,
@p_LoSetpoint31       int,
--------------------------------------------
@p_PeakTemp00  int,
@p_PeakTemp01  int,
.
.
@p_PeakTemp30  int,
@p_PeakTemp31  int,
-------------------------------------------
@p_HiSetpoint00       int,
@p_HiSetpoint01       int,
.
.
@p_HiSetpoint30       int,
@p_HiSetpoint31       int,
--------------------------------------------
@p_MM00 int,
@p_MM01 int,
.
.
@p_MM30 int,
@p_MM31 int,
--------------------------------
@p_DD00 int,
@p_DD01 int,
.
.
@p_DD30 int,
@p_DD31 int,
--------------------------------
@p_YYYY00      int,
@p_YYYY01      int,
.
.
@p_YYYY30      int,
@p_YYYY31      int,
-------------------------------
@p_HH00 int,
@p_HH01 int,
.
.
@p_HH30 int,
@p_HH31 int,
-------------------------------
@p_MIN00       int,
@p_MIN01       int,
.
.
@p_MIN30       int,
@p_MIN31       int,
-------------------------------
@p_SS00 int,
@p_SS01 int,
.
.
@p_SS30 int,
@p_SS31 int,
-------------------------------
@o_Process_Complete    int    OUTPUT
)
AS
SET @o_Process_Complete = 1
```

```
DECLARE @index        int
SET @index = 0
DECLARE @PartIndex    int
DECLARE @LoSetpoint   int
DECLARE @PeakTemp     int
DECLARE @HiSetpoint   int
DECLARE @DateTime     varchar(20)
WHILE (@index<=31)
       BEGIN
              IF (@index = 0)
                     BEGIN
                     SET @PartIndex = @p_PartIndex00
                     SET @LoSetpoint = @p_LoSetpoint00
                     SET @PeakTemp = @p_PeakTemp00
                     SET @HiSetpoint = @p_HiSetpoint00
                     SET @DateTime =
CHAR(39)+CONVERT(VARCHAR(4),@p_YYYY00)+'/'+CONVERT(VARCHAR(2),@p_MM00)+'/'+CONVERT(VARCHAR(2),@p_DD00)+
                     '
'+CONVERT(VARCHAR(2),@p_HH00)+':'+CONVERT(VARCHAR(2),@p_MIN00)+':'+CONVERT(VARCHAR(2),@p_SS00)+CHAR(39)
                     END
              IF (@index = 1)
                     BEGIN
                     SET @PartIndex = @p_PartIndex01
                     SET @LoSetpoint = @p_LoSetpoint01
                     SET @PeakTemp = @p_PeakTemp01
                     SET @HiSetpoint = @p_HiSetpoint01
                     SET @DateTime =
CHAR(39)+CONVERT(VARCHAR(4),@p_YYYY01)+'/'+CONVERT(VARCHAR(2),@p_MM01)+'/'+CONVERT(VARCHAR(2),@p_DD01)+
                     '
'+CONVERT(VARCHAR(2),@p_HH01)+':'+CONVERT(VARCHAR(2),@p_MIN01)+':'+CONVERT(VARCHAR(2),@p_SS01)+CHAR(39)
                     END
.
.
.
.
.

              IF (@index = 31)
                     BEGIN
                     SET @PartIndex = @p_PartIndex31
                     SET @LoSetpoint = @p_LoSetpoint31
                     SET @PeakTemp = @p_PeakTemp31
                     SET @HiSetpoint = @p_HiSetpoint31
                     SET @DateTime =
CHAR(39)+CONVERT(VARCHAR(4),@p_YYYY31)+'/'+CONVERT(VARCHAR(2),@p_MM31)+'/'+CONVERT(VARCHAR(2),@p_DD31)+
                     '
'+CONVERT(VARCHAR(2),@p_HH31)+':'+CONVERT(VARCHAR(2),@p_MIN31)+':'+CONVERT(VARCHAR(2),@p_SS31)+CHAR(39)
                     END
EXEC    ( 'IF NOT EXISTS (SELECT * FROM TempTrackBuffer
                     WHERE PartIndex = '+@PartIndex+' AND DateTime = '+@DateTime+')
                            BEGIN
                                   INSERT INTO TempTrackBuffer
                                   (
                                   Machine_ID,
                                   PartIndex,
                                   LoSetpoint,
                                   PeakTemp,
                                   HiSetpoint,
                                   DateTime
                                   )
                                   VALUES
                                   ('
                                   +@p_Machine_ID+','
                                   +@PartIndex+','
                                   +@LoSetpoint+','
                                   +@PeakTemp+','
                                   +@HiSetpoint +','
                                   +@DateTime +
                                   ')
                            END
       ')
       SELECT @index = @index+1
              IF @index = 32
BREAK
              ELSE
CONTINUE
END
SET @o_Process_Complete = 0
GO
```

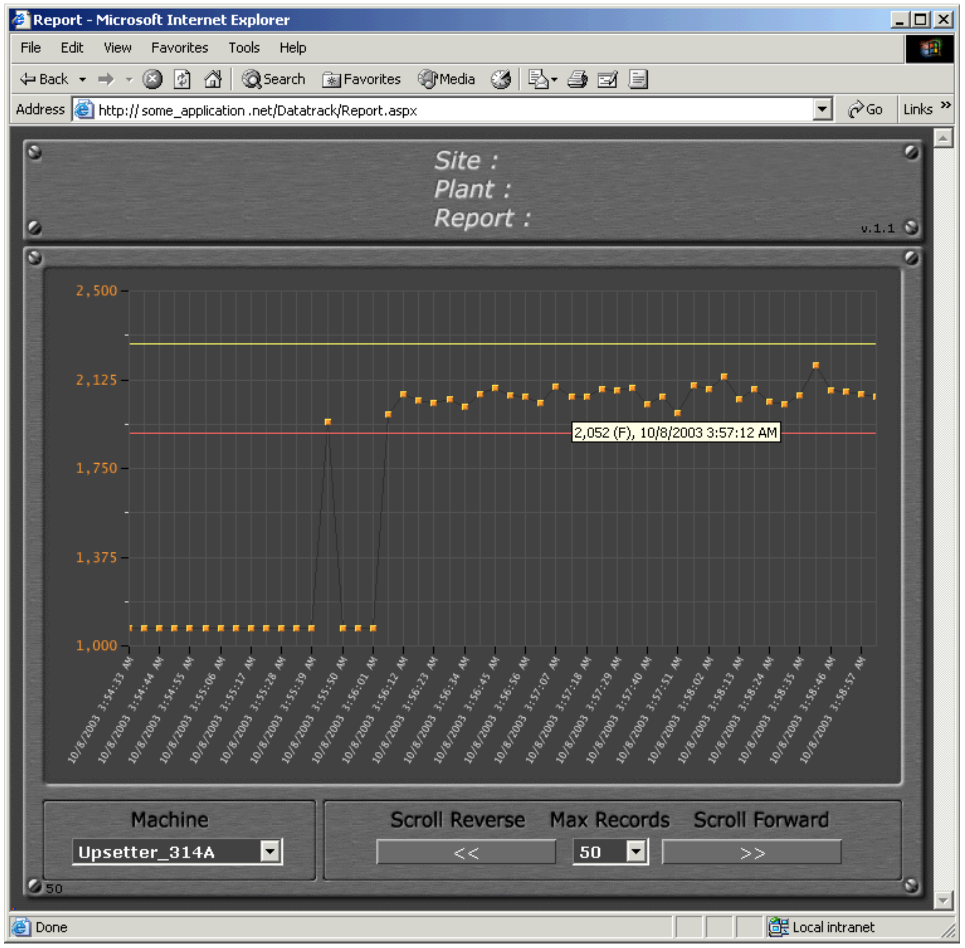
Attachment C. ***Data separator T-SQL stored procedure (Scheduled Job). Code example:***

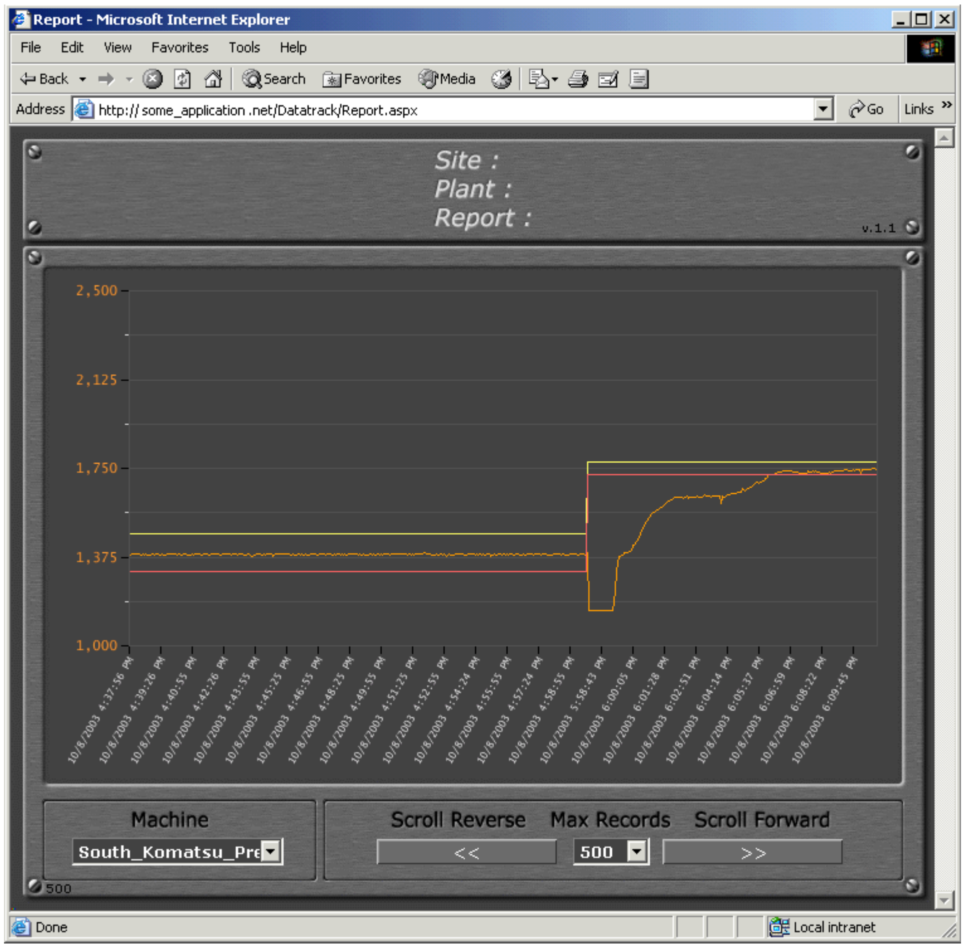
```
CREATE PROCEDURE sp_DataDump

AS

DECLARE @Index INT
DECLARE @p_MachTableName VARCHAR(255)
DECLARE @p_Id VARCHAR(3)
DECLARE @SELcnt INT

SET @p_Id = 0

--GET INDEX COUNT OF TempTrackBuffer DATABASE EXCLUDING DUPS
SET @Index = (SELECT COUNT (DISTINCT MACHINE_ID)
FROM [dbo].[TempTrackBuffer])
--END INDEX COUNT

--BEGIN DATA COLLECTION LOOP UNTIL ALL DISTINCT RECORDS ARE COLLECTED
WHILE (@Index<>0)
BEGIN
--SET MACHINE TABLE ID BY MACHINE_ID IN TempTrackBuffer TABLE MOVING 1 ROW AT A TIME EXCLUDING DUPS
SET @p_Id = (SELECT DISTINCT TOP 1 CONVERT (VARCHAR(3), MACHINE_ID)
FROM [dbo].[TempTrackBuffer]
WHERE MACHINE_ID<>@p_Id)
--END NAMING

--GET TABLE NAME FROM FIS_MACHINE_REF TABLE USING TABLE ID AND REPLACE ANY SPACES WITH _
SET @p_MachTableName = (SELECT REPLACE (DESCRIPTION_M, ' ', '_')
FROM [dbo].[FIS_Machine_Ref]
WHERE Machine_Id_M = @p_Id)

--GET SELECT COUNT
SET @SELcnt = (SELECT COUNT (*)
FROM [dbo].[TempTrackBuffer]
WHERE MACHINE_ID = @p_Id) - 32

--IF MACHINE TABLE DOES NOT EXIST CREATE IT
If not exists (select * from dbo.sysobjects where id = object_id(N'[dbo].['+@p_MachTableName+']') and
OBJECTPROPERTY(id, N'IsUserTable') = 1)
BEGIN
EXEC
('
BEGIN
CREATE TABLE [dbo].['+@p_MachTableName+'] (
        [Record_ID] [int] NOT NULL ,
        [PartIndex] [int] NULL ,
        [Machine_ID] [int] NULL ,
        [LoSetpoint] [int] NULL ,
        [PeakTemp] [int] NULL ,
        [HiSetpoint] [int] NULL ,
        [DateTime] [datetime] NULL
) ON [PRIMARY]
END
')
--UPDATE TABLE INDEX WITH NEW TABLE NAME
INSERT INTO [dbo].[Table_Index]
VALUES (@P_MachTableName)

END
--END CREATE TABLE IF NOT EXISTS

--COPY DATA FROM TempTrackBuffer TABLE TO MACHINE TABLE AND CLEAN UP TempTrackBuffer
IF @SELcnt > 32
BEGIN
EXEC
('
BEGIN
INSERT INTO [dbo].['+@p_MachTableName+']
SELECT TOP '+@SELcnt+' *
FROM [dbo].[TempTrackBuffer]
WHERE MACHINE_ID = '+@p_Id+'
ORDER BY DateTime asc

DELETE FROM [dbo].[TempTrackBuffer]
WHERE PartIndex IN (SELECT TOP '+@SELcnt+' PartIndex
FROM [dbo].[TempTrackBuffer]
WHERE MACHINE_ID = '+@p_Id+'
ORDER BY DateTime asc)
END
')
END
--END COPY AND CLEAN

--COUNT DOWN
SELECT @Index = @Index-1
IF @Index=0
BREAK
ELSE
CONTINUE
END
GO
```

Attachment D. ***Data retrieval T-SQL stored procedure. Code example:***

```
SET ANSI_NULLS ON
SET ANSI_WARNINGS ON
DROP PROC sp_ScrollData
GO

CREATE PROCEDURE sp_ScrollData
(
@p_TableName   VARCHAR(255),
@p_Size        int,           --50,500 or 1000
@p_DateTimeRef datetime,
@p_Direction   varchar(3)     --FWD, RWD
)
AS
-- Declare service variables
       DECLARE @q VARCHAR(1)
       SET @q = CHAR(39)             -- This is quote '
       DECLARE @v_DateTime VARCHAR(22)
       SET @v_DateTime = CHAR(39)+''+Convert(Varchar(22),@p_DateTimeRef,120)+''+CHAR(39)

IF @p_Direction = 'Rew' -- Select previous n records << REW
       EXEC ('SELECT a.* FROM
       OPENROWSET('+@q+'SQLOLEDB'+@q+','+@q+'DETNTFISRPT1'+@q+';'+@q+'<<login>>'+@q+';'+@q+'<<password>>'+@q+',
       '+@q+'SELECT top '+@p_Size+' [DateTime],LoSetpoint,PeakTemp,HiSetpoint from [DF_Heat_Track].dbo.
['+@p_TableName+']
       WHERE [DateTime] <= '''+@v_DateTime+'''
       ORDER BY [DateTime] DESC'+@q+') AS a
       ORDER BY a.[DateTime]')
ELSE                  -- Select next n records >>FWD
       EXEC ('
       SELECT TOP '+@p_Size+' [DateTime],LoSetpoint,PeakTemp,HiSetpoint FROM .dbo.['+@p_TableName+']
       WHERE [DateTime] >= '+@v_DateTime+'
       ORDER BY [DateTime] ASC')

GO
```

Attachment E. ***Report examples:***


Report - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http:// some_application .net/Datatrack/Report.aspx
Go Links
Site :
Plant :
Report :
v.1.1
2,500
2,125
1,750
1,375
1,000
2,052 (F), 10/8/2003 3:57:12 AM
10/8/2003 3:54:33 AM
10/8/2003 3:54:44 AM
10/8/2003 3:54:55 AM
10/8/2003 3:55:06 AM
10/8/2003 3:55:17 AM
10/8/2003 3:55:28 AM
10/8/2003 3:55:39 AM
10/8/2003 3:55:50 AM
10/8/2003 3:56:01 AM
10/8/2003 3:56:12 AM
10/8/2003 3:56:23 AM
10/8/2003 3:56:34 AM
10/8/2003 3:56:45 AM
10/8/2003 3:56:56 AM
10/8/2003 3:57:07 AM
10/8/2003 3:57:18 AM
10/8/2003 3:57:29 AM
10/8/2003 3:57:40 AM
10/8/2003 3:57:51 AM
10/8/2003 3:58:02 AM
10/8/2003 3:58:13 AM
10/8/2003 3:58:24 AM
10/8/2003 3:58:35 AM
10/8/2003 3:58:46 AM
10/8/2003 3:58:57 AM
Machine
Upsetter_314A
Scroll Reverse
<<
Max Records
50
Scroll Forward
>>
50
Done
Local intranet


Report - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http:// some_application .net/Datatrack/Report.aspx
Go Links
Site :
Plant :
Report :
v.1.1
2,500
2,125
1,750
1,375
1,000
10/8/2003 4:37:56 PM
10/8/2003 4:39:26 PM
10/8/2003 4:40:55 PM
10/8/2003 4:42:26 PM
10/8/2003 4:43:55 PM
10/8/2003 4:45:25 PM
10/8/2003 4:46:55 PM
10/8/2003 4:48:25 PM
10/8/2003 4:49:55 PM
10/8/2003 4:51:25 PM
10/8/2003 4:52:55 PM
10/8/2003 4:54:24 PM
10/8/2003 4:55:55 PM
10/8/2003 4:57:24 PM
10/8/2003 4:58:55 PM
10/8/2003 5:58:43 PM
10/8/2003 6:00:05 PM
10/8/2003 6:01:28 PM
10/8/2003 6:02:51 PM
10/8/2003 6:04:14 PM
10/8/2003 6:05:37 PM
10/8/2003 6:06:59 PM
10/8/2003 6:08:22 PM
10/8/2003 6:09:45 PM
Machine
South_Komatsu_Pre
Scroll Reverse
<<
Max Records
500
Scroll Forward
>>
500
Done
Local intranet