Introduction

This chapter discusses an important topic, moving of data and large data manipulation programs. This includes batching applications as well as the labs at the end of the chapter, Simon Says and Whack-a-Mole.

Modicon, a rival PLC manufacturer to A-B and Siemens, used three simple instructions with capabilities to move data from tables to a fixed location, from a fixed location to a table and from a location in a table to the same location in a second table. These instructions were called:

- Table to Register
- Register to Table
- Table to Table

All PLC manufacturers have instructions with capabilities similar to the Modicon instructions. A-B has a number of different instructions for moving data from a table. The SLC and PLC/5 instruction set used two different techniques for moving data. Siemens uses an indexer and matrices. The ControLogix language uses matrices and indexing as well. While RSLogix5000 looks very similar to RSLogix 500, the tag addressing scheme is very different as well as the instructions that perform indexes and moving data using indexing.

In this chapter, we will attempt to discuss the different indexing instructions presently employed by A-B and Siemens as well as look back at the older indexing instructions used by A-B with their SLC and PLC-5 processor lines. In addition, we will look at a specification from ISA called SP88 and some of its methodologies as to how to build a successful batching application.

SP88 will only be discussed here as to how it allows the PLC programmer to build an application that has the elements needed by the application but in a somewhat standardized way.

Batching systems vary in type and are pictured in the next two figures. The first shows a simple batch system with various ingredients laid on the transfer belt at programmed rates. The system here does not include steps but only the process to mix a number of ingredients. More complex systems include steps after which various activities may occur such as agitation or heating. With the second figure, this is the case. Various ingredients can be added to a weigh mix tank a step at a time. This batch system requires the step architecture discussed in this chapter. Both require recipes. There are advantages to both type. Different processes favor one over the other and some processes can use both. Obviously the transfer belt does not allow for wet ingredients. We will discuss the software needed to step through a recipe and the processes to program a batch with the PLC.
The three figures here show a variety of batch systems. The batch system at left was made by Capstone students at the University of Toledo. It contains three ingredient tanks and a mix tank. Below the mix tank is a weigh cell. Ingredients are added to the mix tank one at a time and then the combined mix is drained from the mix tank.

Lower left is a similar system from industry except the ingredients are solids. Each ingredient is added to the mix tank through an auger arrangement. Each ingredient tank is shown with vibration equipment that guarantee an even flow of solid into the auger.

Lower right is a mix system with ingredients simultaneously laying a weighed amount on the belt at a controlled rate. There is no mix tank in this system. The ingredients are layered onto the belt and then fed into a melter or other process.

Data Moving - an Example

We will be using an example of a simple batching system to discuss indexing and moving of data and construction of batching systems. The batch we discuss consists of:

15 recipes for making, as an example, pancake batter:
Each recipe consists of up to 16 steps
Each step consists of 4 integer words
Each recipe is given a 4-digit identifier number 0001-9999 that the operator may enter into a given location prior to requesting that the batch be made
When the recipe is moved into a batch of batter, the recipe is moved to an active recipe area. With the start of the recipe, steps are moved one by one from the active recipe area to the active step area.

As a step is executed, information about the step is collected. For instance, the actual weight added is found and added to the information already found in the step.

As a step is made, the information about the step including the actual data is saved in a 6 integer word group with a time stamp in an output table. As a new recipe starts, the time stamp at the start of the recipe is saved with the recipe number. Then the step information is saved step by step.

To selectively move a recipe from storage to the active area, with A-B processors, the COP command is selected. Moving of large portions of data at a time is referred to as a Copy move. Many times the computer supervising the process will store the many recipes in the computer memory and copying these recipes into the PLC. An alternative technique would provide the PLC with only the recipe presently being made and the next recipe to be made leaving only the two recipes in the PLC’s memory. In general, it is advisable to use the database capabilities of the computer to control movement of recipes rather than storing and moving large portions of data in the PLC and occupying the PLCs memory with recipes that are rarely if ever used.
COP is used for large data table moves:

Contacts that are usually defined as one-shots are selecting large data blocks to be moved. In the example, B3:2/1, B3:2/2, B3:2/3 are selectively picking recipe 1, 2, or 3 to move to the active region to make one of the three recipes the active recipe. The active recipe storage area is in area N7 starting at word 10 and having length of 64 words.

Fig. 13-1  A-B Copy Commands moving large amounts of data
Moving Small Amounts of Data:

Indexed Addressing is used to selectively move a step from the active recipe to the active step. If B3:2/5 is energized (one-shot), the words from the active recipe are moved into locations for use in the active step. The first word selected moves to N7:30. The second word moves to N7:31. The third moves to N7:32 and the fourth to N7:33.
As the batch program moves through steps of the recipe, the indexing pointer moves starting at N7:10 and moves through the recipe.

MOV instructions are used for moving smaller amounts of data selectively using indexed or indirect addressing. Single word MOV statements are used to select individual word groups.

Indexing will be discussed later. For the SLC processors, the index is stored in location S:24. For Siemens and A-B Compact/Control Logix processors, the index is a word used in the move instruction. Matrices replace fixed locations for both Logix and Siemens processors.

The following shows words being moved to an active storage area from the steps in a recipe as the program moves through the steps. First the words of step are moved to the active area. This is also referred to as the work area or the register area. Next step two data is moved, then step three, etc. As each step is moved into the active area, a sequence of programming statements is executed allowing the program to execute the entire step and then move on. The sequence of the program can easily be displayed in a state diagram. The state diagram may have many parts or be very simple. The state diagram may be as simple as found in Fig. 13-3 or more.
Execution at the step level is accomplished with a state diagram program. Movement through a recipe is accomplished with a set sequence of options that must be addressed for each step.
After a step is executed, data is collected about what actually happened. If a scale is used, the target weight is usually not the actual weight of the step. If the operator has an option to choose an action, the results of the choice are likewise remembered by the collected data. This data forms a step in the Recipe Report for a recipe. The word count may be equal to, more than or less than the active step recipe information.

A more ordered approach would be to allow the A-B database handler RSSQL handle the data from recipes. When requested, RSSQL would hand a recipe to the active recipe residing in storage in the PLC. After an active step is executed, RSSQL hands the data back to a second
Next, we will look at the instructions for moving small and large packets of data in the PLC. The Siemens instruction set will be discussed first with emphasis on moving data based on an indexer. Then the A-B commands will be discussed based on the RSLogix 5000 instruction set. A discussion will follow that shows the older SLC and PLC-5 instructions used for moving data. These older methods were used in the above examples. The pointer in these examples is the number found in S:24. As this number is manipulated, the pointer is moved down the table. The S:24 value is referred to as the indexer. The rules for indexing with this method are referred to as indexed addressing. The method was discontinued with the RSLogix 5000 instruction set. Now, the indexing is accomplished with a matrix and the index is the number in the matrix [ ] location. Matrices are also used by Siemens. With each method, the index value must be controlled or the program will have problems and probably cause the processor to fault. A pointer out of range is one of the first problems to look for with batching programs if the processor faults.
Creating Arrays

Creating arrays in Siemens and Allen-Bradley is not complex. For Siemens, follow the following two figures to create an array. Note that arrays are not available in the M Table.

To create arrays in Siemens PLCs, use the Data Block type for creating a storage array. First, choose the "Add new block" command and select Global DB. You will then be able to give information that defines the array desired. Length and data types are necessary. For most applications, the data type is INT or DINT. You may choose to have multiple arrays for multiple data types associated with a list of variables. You may also designate a UDT for this purpose.

The array created in the example above shows an array Array with 51 integer values.
Allen-Bradley allows creation of an array with creation of a named tag and then choosing the data type and dimension. There is no M table with Allen-Bradley and the type and length is chosen below:

The array length is determined in the boxes below. Multiple array dimensions may be chosen as well.

Fig. 13-5 Matrix defined in A-B’s Tag Table
Siemens File Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE</td>
<td>Move value</td>
</tr>
<tr>
<td>FieldRead</td>
<td>Read field</td>
</tr>
<tr>
<td>FieldWrite</td>
<td>Write field</td>
</tr>
<tr>
<td>MOVE_BLK</td>
<td>Move block</td>
</tr>
<tr>
<td>UMOVE_BlK</td>
<td>Move block uninterruptible</td>
</tr>
<tr>
<td>FILL_BLK</td>
<td>Fill block</td>
</tr>
<tr>
<td>UFILL_BLK</td>
<td>Fill block uninterruptible</td>
</tr>
<tr>
<td>SWAP</td>
<td>Swap</td>
</tr>
</tbody>
</table>

The Move instructions have been redesigned from earlier versions and the FieldRead and FieldWrite instructions are now listed as legacy instructions (not to be used in future). The following array move is used for the array to working word discussed above:

Fig. 13-7 Siemens Move Using Array
A-B Data Handling in ControLogix PLCs

Examples of the types of addressing available in the ControLogix processor will demonstrate addressing used in the newer A-B processors. In the following, rung 2 turning on a coil out1 is not as simple as it first may appear. A table table1 is entered in the database seen below. The table consists of 320 bits. Two separate pointers determine the status of a particular bit. Index1 may vary from 0 to 9 determining the word in the table to view. Index2 may vary between 0 and 31. The value of index2 determines the bit in the word being viewed.

Consider the table expanded further as table2[index1,index2],[index3]. The index capabilities of arrays significantly enhance the power of the PLC to perform complicated operations in an orderly manner.
The following array in rung 3 uses a toggle bit to execute an ADD block. In this example, the same 10 word table is used to selectively add one to a second word in the table.

For instance, if index1 = 1 and index2 = 1, the instruction would add 1 to table1[1]. If index1 = 3 and index2 = 4, the contents of table1[4] would be updated with the contents of table1[3] + 1. For-Next Loops are not traditionally included in ladder logic for the reason that a scan can appreciably be lengthened if any loop is executed. For-Next Loops provide a looping control mechanism that is very useful but is capable of lengthening the loop execution time. They have been included in the PLC-5 but not in the SLC ladder instruction set. They are also included in the RSLogix 5000 programming software for both the Control Logix processors as well as the Compact Logix processors. Excluded from the Control Logix and Compact Logix processors, however, are the two addressing modes: indexed and indirect.

The For-Next loop requires an indexer that increments each pass the program makes through a subroutine. The subroutine becomes the program used to execute the logic of the For-Next operation.
In the above example, the routine MainRoutine executes. Each scan that tag_1 energizes, the FOR loop Sub_z executes. The program Sub_z executes 10 times with the value of int_point incremented by 1 from 0 to 9.

Of course, Sub_z allowed to call a subroutine as well with the effect of providing a loop within a loop. The looping procedure is used for table manipulation similar to the MOV and COP commands described in this chapter. If the FOR command is not available, a method using a variable number of scans to execute a subroutine may be employed.

**UDT’s**

Both A-B and Siemens use UDTs for defining variable arrays of data.

A user-defined structure can contain any base data type (e.g., SINT, INT, DINT, BOOL, REAL) or structure (either predefined or user-defined). In addition, a single-dimensional array can be included as a structure member.

To create a user-defined structure, right click on the User-Defined folder in the Controller Organizer, and choose New Data Type. The Data Type editor will appear, from which you can define your new data type.

Many control programs require the ability to store blocks of information in tables that can be traversed at runtime. RSLogix 5000 supports this requirement by providing the ability to create custom arrays with up to three separate dimensions (i.e., row, column, and depth). Individual cells within an array may contain any base data type (e.g., SINT, INT, DINT, BOOL, REAL) or structure (either predefined or user-defined).

The example of a UDT is from the database example of a field. The field defines a set of data of different data types and allows entry into a table using these data types. UDTs are the same. An example of a database table ‘employees’ is as follows:
CREATE TABLE employees ( 
  emp_no INT NOT NULL, -- UNSIGNED AUTO_INCREMENT??
birth_date DATE NOT NULL,
first_name VARCHAR(14) NOT NULL,
lst_name VARCHAR(16) NOT NULL,
gender ENUM ('M','F') NOT NULL, -- Enumeration of either 'M' or 'F'
hire_date DATE NOT NULL,
)

Circular Table

One of the file types studied in a programming course that may be useful in the Recipe Report portion of the program is the circular table structure. The last step executed is presented in an area with its time stamp and the computer is assumed to pick up the actual step information and record it. If multiple steps are available for the polling computer, then the circular file structure becomes more important. For instance, the last few steps of information are kept in the file. The PLC program continues to increment through the steps working its way down the recipe and sending the actual step information to the circular table. When the end of the circular table is reached, the PLC starts again at the top of the table. This approach allows steps to be read multiple times by the computer. If the polling computer fails to read a record, the table will continue to fill till the end of the table is reached. Only when the table is full and wraps over old data will data be lost if the polling computer is still unable to read the data.

An example of a circular table:

<table>
<thead>
<tr>
<th>Time Stamp</th>
<th>Step 1 W1</th>
<th>Step 1 W2</th>
<th>Step 1 W3</th>
<th>Step 1 W4</th>
<th>Step 1 W5</th>
<th>Step 1 W6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time Stamp</td>
<td>Step 2 W1</td>
<td>Step 2 W2</td>
<td>Step 2 W3</td>
<td>Step 2 W4</td>
<td>Step 2 W5</td>
<td>Step 2 W6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time Stamp</td>
<td>Step 3 W1</td>
<td>Step 3 W2</td>
<td>Step 3 W3</td>
<td>Step 3 W4</td>
<td>Step 3 W5</td>
<td>Step 3 W6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Empty, yet to be filled in Step
While data may be stored in the PLC, most large databases are kept in a computer closely linked to the PLC. The circular table may be used to buffer data if the computer is not responding to the PLC. A database is useful in organizing large amounts of data. Using a database for applications such as this offloads the PLC from the burden of data handling, giving the PLC the flexibility to do what it is designed to do best - control machines.

**Indirect Addressing for SLC and PLC/5 PLCs (Older Addressing Schemes)**

This addressing format allows a storage location to specify the number in the file of the file, element, or bit in the direct logical address. Up to two address numbers in the direct address are allowed with indirect addresses.

For example:  
#F[N7:4]:0 identifies a floating point file whose number is found in the N7:4 location. If this location contains 99, the 99th file would be accessed as element 0. This would be equal to F99:0.

Rules for using Indirect addressing:

1. Indirect addresses are indicated with brackets

2. Address must be direct logical address with N, T, C, or R types. Type N (Integer) is recommended but not required.

3. # is not to be used inside brackets

4. The element number must be within the file’s length. A fault will occur if not.

5. When used to store file number, the file number must represent the same type as the prefix.

Examples:

B3/[ ], B[ ]/[ ], N7:[ ]/8, C[ ]:5.DN

Example:

The following contact in a program would have various values depending on the value in N7:0.

B3/[N7:0]
Example:

```
  B3:0
  1
```

Fig. 13-11 A-B Indirect Addressing Example from SLC Instruction Set

The example makes the counter variables being added selectable from any of the counters in C5.

Note that addresses are not limited to whole word addresses. Bit addresses may be referenced as well.

```
  B3:0
  [N7:10]
```

Fig. 13-12 A-B Indirect Addressing Example from SLC Instruction Set

The rung references B3/ and then a bit in the B3 table referenced from 0 using N7:10 as that reference. For example, if N7:10 contained the number 30, the bit reference would be B3/30 or B3:1/14.

An editorial comment on Indirect Addressing:

**It is hard to debug or troubleshoot if you are not the initial programmer. So, don’t use it unless you find no alternative means to program the task at hand.**

It is believed that the use of this addressing method to obscure the logic behind the program led the designers to discontinue it with the RSLogix 5000 language. Both the indirect and indexed methods were discontinued with the newer languages.

If one becomes fluent with all the new languages, the procedural STL language may be the best to be used for indexing program development. Although we predominantly use LAD here, the
STL language has many benefits and should be considered when planning a program with indexing in mind.

**Indexed Addressing for SLC and PLC/5 PLCs**

Indexed addressing allows an offset of an address by a number of words stored in location S:24. To identify indexed addressing, place the # symbol immediately before the file-type identifier in the address:

For example:  #N7:0 refers to word 0 of the N7 file offset by the number stored in S:24.

When using Indexed Addressing:

- Use care to insure that the index value (positive or negative) does not exceed file bounds.
- With instructions using two or more indexed addresses in the same instruction, the offset will be the same for all addresses.
- Use care to reset the offset to its desired value before enabling an instruction having indexed addresses.

Example:

The MVM instruction uses indexed addressing as follows:

![Fig. 13-13 A-B Indexed Addressing Example from SLC Instruction Set](image-url)
Using the MOV instruction as a reference, observe that the table of results uses the offset value found in S:24 to compute a new address for the MOV instruction. The example above moves the value in N7:14 to N7:54 when the offset in S:24 = 4. Only one word is moved when the rung is executed. Other program statements should be added to the rung to increment or decrement S:24 to move other locations.

Another example using one offset address and one fixed address:

<table>
<thead>
<tr>
<th>Value</th>
<th>Base</th>
<th>Offset (Value in S:24)</th>
<th>Actual Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>N7:10</td>
<td>4</td>
<td>N7:14</td>
</tr>
<tr>
<td>Destination</td>
<td>N7:50</td>
<td>4</td>
<td>N7:54</td>
</tr>
</tbody>
</table>

This is commonly referred to as a Table to Fixed move. This type of rung is used to get a value sequentially or randomly from a table. It is used in programming recipe routines.
An example using a fixed first address and an indexed second address is included. It is used to get a value from a fixed location into a table. It is used many times to save the status of an event that happened in sequence. Picture a line of cars with numeric only license plates. The output table would save the sequential status these numbers starting with N7:50 and sequentially store the license plates in order as they passed by a point.

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>Offset(S:24)</th>
<th>Actual Dest</th>
</tr>
</thead>
<tbody>
<tr>
<td>N7:0</td>
<td>N7:50</td>
<td>0</td>
<td>N7:50</td>
</tr>
<tr>
<td>N7:0</td>
<td>N7:50</td>
<td>1</td>
<td>N7:51</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 13-15 A-B Indexed Addressing Example from SLC Instruction Set
To enter a program using Indexed Addressing, build a table as follows and enter the program listed below:

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>N7:0</td>
<td>5</td>
</tr>
<tr>
<td>N7:1</td>
<td>10</td>
</tr>
<tr>
<td>N7:2</td>
<td>15</td>
</tr>
<tr>
<td>N7:3</td>
<td>20</td>
</tr>
<tr>
<td>N7:4</td>
<td>25</td>
</tr>
<tr>
<td>N7:5</td>
<td>30</td>
</tr>
<tr>
<td>N7:6</td>
<td>35</td>
</tr>
<tr>
<td>N7:7</td>
<td>40</td>
</tr>
<tr>
<td>N7:8</td>
<td>45</td>
</tr>
</tbody>
</table>

Using the SLC processor to execute a For-Next loop employs a technique that allows a variable number of scans to access the subroutine code. For instance, the following code could accomplish the same function as that shown above if timing constraints were considered.
The program called in U:3 may be used to respond in a manner similar to the For-Next program if the output of the code can wait the number of scans needed to loop through the entire range of numbers in int_point. Care must be taken to initialize N7:10 to 0 prior to execution of the code as well. When the final path through the loop is executed a bit usually is set to signal any user program that data from the loop is available. This is more cumbersome than the program of the Compact Logix processor. It also does not easily support loops within loops. Extreme care must be taken when using this program technique. It is important to note that this type of program control is used by many programmers and the student should be aware of its implementation in existing automation programming.

### Comparing Older AB Addressing Modes

As advances are made from the PLC/5 to SLC to RSLogix 5000 processors, addressing requirements have been enhanced as well. The RSLogix 5000 processors use indexed arrays to provide functional equivalent programming to the indexed and indirect methods of the PLC/5 and SLC processors. Modes may be mixed, causing a number of programming types which may or may not be substituted with the indexed array of RSLogix5000. The following list of comparisons shows some of the evolution from the PLC/5 to SLC to RSLogix5000 addressing.

**Indexed:**

```
#N7:0
```

Supported by PLC/5, SLC
Available in RSLogix5000 by using Indexed Arrays

**Indirect Word:**

```
(N7 : [N7 : 6])
```

Supported by PLC/5, SLC
Available in RSLogix5000 by using Indexed Arrays
Indirect File: \((N[N7:4]:0)\)  
Supported by PLC/5, SLC  
Not available in RSLogix5000

Indexed + Indirect Word: \((#N[N7:2])\)  
Supported by PLC/5, SLC  
Available in RSLogix5000 by using Indexed Arrays

Indexed + Indirect File: \((#N[N7:8]:0)\)  
Supported by PLC/5, SLC  
Not available in RSLogix5000

Indexed + Indirect File + Word: \((#N[N7:3]:[N7:4])\)  
Supported by PLC/5, SLC  
Not available in RSLogix5000

Nested Indirection: \((N7:[N7:[N7:2]])\)  
Supported by PLC/5, SLC  
Not available in RSLogix5000

While PLC/5 and SLC processors use various combinations of Indexed and Indirect pointers to move data, the RSLogix5000 processors use indexed arrays to accomplish the same task.

FOR-NEXT looping is not used in the SLC processor family but is supported by the PLC/5 family. Use of multiple scans is necessary to provide the equivalent functionality to FOR-NEXT looping when using the SLC processors.

**Comparison of MOVE Instructions (with Examples from Siemens and A-B)**

**The Table to Table Move**

The MVM instruction uses indexed addressing as follows:
Siemens’ Move equivalent of the indexed address move above:

```
MOVE_BLK
EN
IN
COUNT
#IN_Array(index1)
1
OUT
ENO
Tag_Out
#OUT_Array(index2)
```

Allen-Bradley’s Move equivalent of the indexed address move above:

```
COP
Copy File
Source
IN_Array(index1)
Dest
Out_Array(index2)
Length
1
```

Notice that both the Siemens and A-B newer move statements allow more than one element to be moved. Also notice that index values for the two arrays may be the same or different.
The Table to Register Move

Another example using one offset address and one fixed address:

<table>
<thead>
<tr>
<th>Source</th>
<th>Offset (S:24)</th>
<th>Actual Source</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>N7:5</td>
<td>0</td>
<td>N7:5</td>
<td>N7:30</td>
</tr>
<tr>
<td>N7:5</td>
<td>1</td>
<td>N7:6</td>
<td>N7:30</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>20</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This is commonly referred to as a Table to Fixed move. This type of rung is used to get a value sequentially or randomly from a table. It is used in programming recipe routines.

Siemens’ Move equivalent of the indexed address move above: (obsoleted and MOV block used now)
Allen-Bradley’s Move equivalent of the indexed address move above:

![Diagram of MOV instruction]

**The Register to Table Move**

An example using a fixed first address and an indexed second address is included. It is used to get a value from a fixed location into a table.

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>Offset(S:24)</th>
<th>Actual Dest</th>
</tr>
</thead>
<tbody>
<tr>
<td>N7:0</td>
<td>N7:50</td>
<td>0</td>
<td>N7:50</td>
</tr>
<tr>
<td>N7:0</td>
<td>N7:50</td>
<td>1</td>
<td>N7:51</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 13-20
Siemens’ Move equivalent of the indexed address move above: (again, obsoleted and MOV block used now)

![FieldWrite EN INDEX VALUE #OUT Array(index) OUT ENO Tag_Out #Fixed_Value Fixed_Value OUT Array(index) MOV Source Fixed_Value Dest OUT Array(index) MOV]

Allen-Bradley’s Move equivalent of the indexed address move above:

**Scale Weighing**

Scale Weighing Systems are an important part of a batch system. Siemens provides a load cell interface system for weighing applications called the SIWAREX WP231. It is pictured in the picture in Fig. below.

![Fig. 13-22]

The electronic weighing system has the following characteristics as listed by Siemens:

“
- Uniform design technology and consistent communication in SIMATIC S7-1200
- Parameter assignment by means of HMI panel or PC
- Uniform configuration option in the SIMATIC TIA Portal
- Measuring of weight with a resolution of up to 4 million divisions
- High accuracy, 3000 d, legal for trade according to OIML R76
- Legal-for-trade display with SIMATIC operator panel or PC
- High measuring rate of 100/120 Hz (effective interference frequency suppression)
- Limit monitoring
- Flexible adaptation to varying requirements
- Easy calibration of the scales using the SIWATOOL program
- Automatic calibration is possible without the need for calibration weights
- Module replacement is possible without recalibrating the scales
- Use in Ex Zone 2 / ATEX approval
- Intrinsically safe load cell supply for Ex Zone 1 (SIWAREX IS option)
- Diagnostics functions”

**SIWATOOL overview**

SIWATOOL does not only offer support when you set the scale but also when you analyze the diagnostic buffer that can be saved after being read out of the module together with the parameters. The display of the current scale status can be configured.
A load cell is pictured in the figure below:

![A load cell is pictured in the figure below.](image)

**Fig. 13-24**

**Fig. 13-25**
A weigh vessel with load cells usually includes four load cells. Some vessels may only include three load cells, however.

Fig. 13-26
There are also load cells located under traveling weigh belts. Systems that weigh moving material include averaging algorithms that weigh the belt and material but tare the weight of the belt out of the weight.

The following weigh vessels show the location of load cells with the vessel suspended in space. These weigh vessels show the traditional locating of load cells at four corners of the tank.

The Siemens system is described in the manual: Weighing systems Electronic weighing system SIWAREX WP231.
Also, the load cell can be terminated in a converter box similar to the Red Lion Strain/Load Cell Panel Meter pictured below. Since it can be used to interface to a PLC using an optional analog output, the Red Lion is used simply to pass through a signal from the scale to the PLC after linearization has occurred with the scale signal.

![Red Lion PAXS0000 Strain/Load Cell Panel Meter](image)

Fig. 13-29
Check List for Writing PLC Batch Application

From the ISA website, the following gives an insight into considerations for the batching application:

“The ISA88 committee has published a series of standards on batch control in industrial automation systems.”

They are:

**ANSI/ISA-88.00.01-2010, Batch Control Part 1: Models and Terminology**

This standard “provides standard models and terminology for defining the control requirements for batch manufacturing plants”

**ANSI/ISA-88.00.02-2001, Batch Control Part 2: Data Structures and Guidelines for Language**

This standard “defines data models that describe batch control as applied in industrial automation systems, data structures for facilitating communications within and between batch control implementations, and language guidelines for representing recipes”

**ANSI/ISA-88.00.03-2003, Batch Control Part 3: General and Site Recipe Models and Representation**

This standard “defines a model for general and site recipes; the activities that describe the use of general and site recipes within a company and across companies; a representation of general and site recipes; and a data model of general and site recipes”

**ANSI/ISA-88.00.04-2006, Batch Control Part 4: Batch Production Records**

This standard “provides a detailed definition for batch production records, establishing a reference model for developing applications for the storage and/or exchange of batch production records. Implementations based upon the standard will allow retrieval, analysis, and reporting of selected batch production record data”

A fifth standard is under development:

**Implementation Models & Terminology for Modular Equipment Control**

The purpose of the SP88-ISA-88 specification is to provide a common strategy for all batch applications programmed. These include PLC programs. To be included are the physical and functional implementations. The functional model includes “the relationships between the five types of control recipe management, scheduling, sequential control, regulatory control, and safety interlock systems.”

Examples are the definitions for modes, states and alarms for a typical batch system. They are
defined in the specification as:

- Auto
- SemiAuto
- Manual
- Bypassed
- Controlled
- Reset
- AlarmPresent
- AlarmAcknowledged

As discussed previously, the reporting function is important in that values are required by the higher level computer. Such values as actual weights by the scale, errors, status of the batch should be saved and reported for each step of the batch.

While the chemical engineer or mechanical engineer is usually in charge of determining the equipment design and feeds for the process, the electrical engineer/program designer is responsible for programming the process. Thus there is a concern and desire to have input in the process from the beginning. The PLC program should be knowledgeable of the feed rates, the ingredients to fill, the coordination of when and how to mix and dump the material, when to apply heat, etc.

With the batching system, scales and other material feed devices are required to monitor and control the feed rates of the various materials. SP-88 defines the following material feed types:

- **FILLING**
  “The SINGLE transfer (movement) of a specified amount of product from one single location to another location”

- **DOSSING**
  “The SINGLE transfer (movement) of a specified amount of product from one location into a continuous process”

- **FORMULATION**
  “MULTIPLE transfers (movements) of specified amounts of products from various locations into a single location”

- **BLENDING**
  “MULTIPLE transfers (movements) of specified amounts of products from various locations into a single location plus a single ADDITIONAL process phase – mixing”

- **BATCHING**
  “MULTIPLE transfers (movements) of specified amounts of products from various locations into a single location plus multiple ADDITIONAL process phases – heating, cooling, wait, mix, agitate, dump etc.”
The different methods of adding material to a batch are discussed. Most involve scales but there are those that involve a meter or level sensor. Methods for measuring material include:

**Material Feed Types**

- Gain In Weight Feeders
- Loss In Weight Feeders
- Flow Meter Feed or Metered Feed
- Dump To Empty
- Hand Add

To add material using any of the above methods, a scale or load cell system should probably be used to accurately weigh the product being added. The scale can be either on the container being added to or fed from. The steps in weighing are critical in that consistency needs to be maintained and proper records kept of all transactions. Decisions need to be made as well whether the weight added is acceptable or not. The SP-88 document divides the weighing cycle into six states as follows:

- Pre Feed
- Feed Start
- Feed
- Feed Stop
- Feed Finish
- Post Feed

It may be important to the programmer to identify every step and build their program around these steps. Certainly the steps add order to the program and aid in troubleshooting the system. The state diagram shown below also helps in identifying areas of concern and giving aid in programming the feed.

**State Diagram of Material Add System**

A state diagram is included in the SP88 diagram for each step of adding material and used as a review of the steps outlined above:
Alarms are likewise included the presence of an alarm, the setpoint over or under-run of the add and whether or not the actual weight was acceptable to the operation before proceeding.

**REPORTS**

The following reports should be considered when weight is added to a mix:

- ActualFeedWeight
- Error
- ExistStatus

**Material Transfer Terminology**

Some of the terms used in the addition of material using scales or rate feeders include:

Weighing Terminology

- Gross
- Net
- Units
- Zero
- Center of Zero
- Under Zero
Material Transfer Terminology (definitions from SP-88)

“Target ( = Setpoint + Spill)
Set Point (= Target – Spill)
Spill (=Target – Set Point) (preact, in-flight, offset, bias)
- Fixed Spill
- Adaptive Spill
- Predictive Adaptive Spill

Fast Feed (coarse feed)
Feed (fine feed, slow feed, dribble feed)

Control Methods
- BASIC Control – single speed transfer control

Dribble Setpoint or Dribble Target or Slow Feed Target(slow-feed setpoint, dribble setpoint)

Flow Rate

Tolerance
Jog (re-dispensing)
- Manual Jog – operator starts jog feed and operator ends jog feed
- Semi-Automatic Jog – operator acks out of tol, starts jog feed and controller ends jog feed
- Automatic Jog – operator acks selects auto accept out tol, controller starts jog feed and ends jog feed
- Setpoint Error, Target Error, Feed Error”

The inclusion of this kind of information regarding the addition of a material to a batch may seem too much to digest but when faced with the task of programming the entire batch system, many times from no prior program, it is important to ask the right questions and be able to make the system perform to specification. Many people are depending on the accuracy of the batching system since the product being made must guarantee accuracy within specifications on the label of the product in the store. For pancake mix, this is very important, especially for those of us who like good pancakes.
For a simple batching system with one main mix tank

Now that we have somewhat of an idea about a batching system structure, we could picture the instructions used by the various PLC’s to provide the step move to an active step. The steps are activated sequentially starting with step 1, step 2, …

Fig. 13-31
How Would You Program This?

The following picture is of a liquid batching system capable of simultaneously mixing batches in three main mix tanks. In addition, there are six pre-mix tanks – any of which can be mixing a pre-mix for any one of the main mix batches. Each tank measures addition of weight via scales.

Multiple pre-mixes may be required for each main mix and they mix concurrently in various pre-mix tanks. They then add to the main mix at a prescribed time.

Begin to plan the programming of how a recipe is to be constructed to handle such a system as this.

![Liquid Batch System](image)

This is left as a problem for discussion.
Other Types of Recipes

There are other recipe types not necessarily using scales and mx tanks. The following two examples give some insight into the diversity of types of recipes.

Valve Nest:

This example requires a number of paths through a number of valves in a valve nest. Each valve has two limit switches, one closed with a closed valve and one closed with an open valve. Each valve in the transfer path must be set appropriately and its limit switches closed or opened in a recipe in order to allow a transfer to occur. With a number of different transfer paths, these paths may be considered recipes.

Steel Heat/Soak Cycle:

The heating of steel in the hardening process requires a recipe of time and heat setpoint. The hardness of a piece of steel is determined with a recipe.

Several pieces can be inserted in the oven at the same time if they all support the same setpoints and soak times. If various types are to be introduced at the same time, then a check must be performed to determine whether the different recipes can be used at the same time.

This is similar to baking a cake. If one is determined to use the oven to bake a large number of cakes, if one cake is ready to take out, then another cake can be inserted in the oven even if there is a longer baking cake still in the oven.

The rules in the steel example are used to determine whether multiple pieces can be inserted at the same time. These rules can be intricate.
Summary

This chapter explains a number of data handling instructions and provides applications for their use in factory automation. Instructions are provided for a number of operations that would otherwise have required a significant programming effort to provide. Included are the queuing operations using FIFO instructions. Other interesting instructions in data handling include the bit-shift instructions. These instructions are used for shift register part tracking. Instructions used in data manipulation were shown with examples included for each type of instruction. An example of a batching application demonstrated the handling of large amounts of data. The chapter concluded with a discussion of For-Next loops and their inclusion in possible data-handling operations.

Included at the end of the chapter are two important labs using file manipulation to control simple games. The games Simon Says and Whack-a-Mole are used to provide an experience in the use of table-to-working register functions. In the MicroLogix 1000 processor, the only available mode to consider table-to-register moves is the indexed mode. Whether using the MicroLogix 1000 or other processor, these labs provide valuable experience programming simple batching applications such as these. The example program was written for the MicroLogix 1000 processor.

You may be curious how two games, Simon Says and Whack-a-Mole have anything to do with batch applications. If run as sequences of numbers, the same operations used in these games are used for batch applications. The games are to be programmed in this way. They are not to be programmed using random number generation but rather as storage of information in tables and then retrieving the information as the game is being played. The Whack-a-Mole game has the added similarity to a batching program in that one of the options asks for a report back as to how soon after the mole popped up that the button was pushed. This is similar to the report of actual weight added for the batch report. Hopefully these labs will give some insight into actual batch programs.
Lab 13.1  Simon Says

Use four illuminated push buttons to build the game Simon Says.
Use a fifth button to start the game.
Use a sixth light to signal the game is done good.
Use a seventh light to signal the game is done bad.

Simon Says is a sequential game that plays a four-note tune. This lab has no sound so lights will have to do. After the start button is pushed, one light at a time is turned on. The player must mimic or push the button attached to the light. First one light is lit, then two, then three, etc., until a final number is attained. Assume 15 is the final number of steps in the game. The steps are repeated each time the game is played. The values are to be stored in a table and re-used.

Lab 13.1A Add a three-position switch to allow for easy, middle, hard ranges varying the number of steps from 10 to 15 to 20.

Lab 13.1B Automatically rotate through 3 different sets of conditions for each range – easy, middle, and hard.

Lab 13.1C Vary the speed that the lights are display as the player gets closer to the last step.

Lab 13.1D Create a teach mode using a separate button to teach the game a sequence of steps.

Use the program starting on the next page as a suggested beginning point for your logic:

The program below is programmed for three, not four buttons and is programmed using indexed addressing from the SLC architecture PLCs from A-B.
Fig. 13-34a Simon Says Game Program in SLC
Fig. 13-34b  Simon Says Game Program in SLC
Fig. 13-34c Simon Says Game Program in SLC
In addition to the program, a table must be filled in starting at N7:0 in the N7 file.

![Table Data](image)

**Fig. 13-35  Simon Says Game Program in SLC**

The file must also be inserted for the game to work correctly. Set the Radix to *Binary* and set each word from N7:0 to N7: with either a 1 in the bit 0, 1, 2 or 3 position. The game will be played using these entries.
Lab 13.2  Whack-a-Mole

Design a Whack-a-Mole game using only 4 lights. The game is to react to the light by pushing that particular button before the light turns off. Construct the game so that the time between lights is pseudo-random (you pick various numbers) and the next light to turn on is pseudo-random (again, you pick). Count the steps (light turn-ons) and stop at 30. If the person playing the game is successful 10 of the 30 times, blink all the lights a number of times. The steps are repeated each time the game is played. The values for buttons and time delays are to be stored in a table and re-used.

Use a button not part of the game to start the game.

The layout is as follows:

![Whack-a-Mole buttons](image)

Button used to start game

**Lab 13.2A**  Automatically rotate through 3 different sets of data tables.

**Lab 13.2B**  Create a teach mode using a separate button to teach the game a sequence of steps which can then be played.

**Lab 13.2C**  Add a table of results including whether the player hit the light while the light was on and how long the response was delayed from when the light first turned on. Results for each hit are to be saved sequentially in the table.
Exercises:

1. Write logic to move sequential data from a table starting at N7:20 to location B3:20. Use B3/9 to reset to the top of the file (B3/9 is a one-shot pulse). Use B3/10 to move the data (B3/10 is also a one-shot pulse).

   Use the instructions below to fill in changes in the table:

<table>
<thead>
<tr>
<th>Source</th>
<th>Value</th>
<th>Destination</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>N7:0</td>
<td>102</td>
<td>N10:0</td>
<td>0</td>
</tr>
<tr>
<td>N7:1</td>
<td>115</td>
<td>N10:1</td>
<td>0</td>
</tr>
<tr>
<td>N7:2</td>
<td>210</td>
<td>N10:2</td>
<td>0</td>
</tr>
<tr>
<td>N7:3</td>
<td>365</td>
<td>N10:3</td>
<td>0</td>
</tr>
<tr>
<td>N7:4</td>
<td>542</td>
<td>N10:4</td>
<td>0</td>
</tr>
<tr>
<td>N7:5</td>
<td>9</td>
<td>N10:5</td>
<td>0</td>
</tr>
</tbody>
</table>

2. 
\[
\text{MOV} \\
\#N7:0 \\
\#N10:0
\]
\[
\text{and } s:24 = 2
\]

<table>
<thead>
<tr>
<th>Destination</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>N10:0</td>
<td></td>
</tr>
<tr>
<td>N10:1</td>
<td></td>
</tr>
<tr>
<td>N10:2</td>
<td></td>
</tr>
<tr>
<td>N10:3</td>
<td></td>
</tr>
<tr>
<td>N10:4</td>
<td></td>
</tr>
<tr>
<td>N10:5</td>
<td></td>
</tr>
</tbody>
</table>

3. 
\[
\text{MOV} \\
\#N7:0 \\
\#N10:0
\]
\[
\text{and } s:24 = 4
\]

<table>
<thead>
<tr>
<th>Destination</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>N10:0</td>
<td></td>
</tr>
<tr>
<td>N10:1</td>
<td></td>
</tr>
<tr>
<td>N10:2</td>
<td></td>
</tr>
<tr>
<td>N10:3</td>
<td></td>
</tr>
<tr>
<td>N10:4</td>
<td></td>
</tr>
<tr>
<td>N10:5</td>
<td></td>
</tr>
</tbody>
</table>

4. 
\[
\text{MOV} \\
N7:0 \\
\#N10:0
\]
\[
\text{and } s:24 = 3
\]

<table>
<thead>
<tr>
<th>Destination</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>N10:0</td>
<td></td>
</tr>
<tr>
<td>N10:1</td>
<td></td>
</tr>
<tr>
<td>N10:2</td>
<td></td>
</tr>
<tr>
<td>N10:3</td>
<td></td>
</tr>
<tr>
<td>N10:4</td>
<td></td>
</tr>
<tr>
<td>N10:5</td>
<td></td>
</tr>
</tbody>
</table>

5. With the problems 2-4 above, rewrite the logic using Siemens’ LAD equivalent instruction, Allen-Bradley’s RSLogix 5000 LAD equivalent instruction.


7. Review the following from the Chemicals & Petrochemicals Plant Automation Congress 2015 presentation “Batch Process Control Strategy”
8. Review a scale input method and comment on the interface between the scale system and the PLC.

9. Discuss implementation strategies for programming the batch project in Fig. 13-32.
Appendix

**Linking PLC UDT Tags to HMI Faceplates and Pop-ups in TIA Portal V13 SP1 (DMC Corp)**

A blog posted by Jason Mayes in Front Page, PLC, Automation, Siemens PLC, HMI and SCADA, WinCC

PLEASE NOTE: This blog was written using features available in TIA Portal V13 SP1 Update 1. Some of the functionality shown was removed in Update 3, specifically the ability to multiplex UDTs. Hopefully the rest of the information will be still be helpful to you!

Here at DMC, we spend a lot of time programming PLCs and HMIs. While we program systems of all types and flavors, I'm personally most experienced with Siemens (TIA Portal) and Rockwell (RSLogix5000/FactoryTalk View). There are a few new features included in TIA Portal V13 SP1 that I have found to be incredibly useful in the past few months and have allowed me to be even more efficient in my programming. I'm going to focus today on a series of updates that were added in SP1 that make it even simpler to take advantage of the nested data structures we can create in our Data Blocks - specifically, the ability to share UDTs between a PLC and HMI, link UDTs to Faceplates, multiplex arrays of UDTs, and create simple, powerful pop-ups.

Before we get started, let me say that if you're not already taking advantage of PLC data types and Global DBs to build rich data structures in your project, you're missing out. Having come from an object-oriented programming background, I really appreciate what the Function Block/Data Block paradigm will allow you to do in a PLC - it really does push PLC programming towards object-oriented programming. And let's face it, the less time we all spend on repeating tasks and minutiae, the more time we can spend actually programming.

As an exercise, let's pretend we have a project with a large number of simple valves. Of course we like to save time, so we're going to develop a special FB to handle our valves: auto/manual control, alarms, etc. In addition to eliminating the amount of code to be written, this allows us the additional advantage of keeping our future valve logic updates to a single place. Our FB has a few inputs including the feedback signals for open/close, an in-out for our 'valve' data type, and an output for the open/close command.
So, what is in our 'valve' data type? For this simple example, let's create the following PLC data type, **udtValve**:

<table>
<thead>
<tr>
<th>Name</th>
<th>Data type</th>
<th>Default value</th>
<th>Accessible</th>
<th>Visible</th>
<th>Setpoint</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>NormallyOpen</td>
<td>Bool</td>
<td>false</td>
<td></td>
<td></td>
<td></td>
<td>0 = NC, 1 = NO</td>
</tr>
<tr>
<td>InOpenFeedback</td>
<td>Bool</td>
<td>false</td>
<td></td>
<td></td>
<td></td>
<td>0 = no open feedback</td>
</tr>
<tr>
<td>InClosedFeedback</td>
<td>Bool</td>
<td>false</td>
<td></td>
<td></td>
<td></td>
<td>0 = no closed feedback</td>
</tr>
<tr>
<td>TransitionTime</td>
<td>Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>value response time before fault</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Status</td>
<td>Struct</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMDMode</td>
<td>Bool</td>
<td>false</td>
<td></td>
<td></td>
<td></td>
<td>0 = valve is in auto, 1 = valve is manual</td>
</tr>
<tr>
<td>State</td>
<td>Uint</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>0 = indeterminate, 1 = closed, 2 = open</td>
</tr>
<tr>
<td>IsClosed</td>
<td>Bool</td>
<td>false</td>
<td></td>
<td></td>
<td></td>
<td>1 = valve is open (no feedback, will tell expected state)</td>
</tr>
<tr>
<td>IsFaulted</td>
<td>Bool</td>
<td>false</td>
<td></td>
<td></td>
<td></td>
<td>1 = valve is closed (no feedback, will tell expected state)</td>
</tr>
<tr>
<td>Control</td>
<td>Struct</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AutoCMD</td>
<td>Bool</td>
<td>false</td>
<td></td>
<td></td>
<td></td>
<td>1 = actuate valve</td>
</tr>
<tr>
<td>ManCMD</td>
<td>Bool</td>
<td>false</td>
<td></td>
<td></td>
<td></td>
<td>1 = actuate valve</td>
</tr>
<tr>
<td>ResetCMD</td>
<td>Bool</td>
<td>false</td>
<td></td>
<td></td>
<td></td>
<td>1 = reset valve</td>
</tr>
<tr>
<td>CMDModeRequest</td>
<td>Bool</td>
<td>false</td>
<td></td>
<td></td>
<td></td>
<td>0 = request auto mode, 1 = request manual mode</td>
</tr>
</tbody>
</table>

Siemens FB for udtValve

We've broken our data type into three structures: configuration, status, and control. The configuration group of parameters defines the valve's physical behavior and its unique name. The status group contains all status information about the valve and is used both within the PLC and the HMI. The control group has the open/close and fault reset requests. I find breaking things up like this makes it much easier to find the right tag you're looking for when programming. For our valve Function Block, **fbValve**, let's say we have something like this:

**Network 1: Determine valve operating mode**

```plaintext
#Valve.Status.CMDMode

#Valve.Status.CMDMode

#Valve.Control.CMDModeRequest

#ManualModeEnabled

#Valve.Status.CMDMode
```

**Network 2: Determine if valve should be actuated**

```plaintext
#Valve.Control.AutoCMD

#Valve.Status.CMDMode

#ActuateValve

#Valve.Control.ManCMD

#Valve.Status.CMDMode
```
**Network 3:** Check for alarmed state: Valve Close

Should be closed and is not, or should not be closed and is.

**Network 4:** Check for alarmed state: Valve open

Should be open and is not, or should not be open and is.
Now that we’ve developed a re-usable function block for our valves, let’s create a DB to contain all of our valve data. I’m going to create a Global DB and name it `dbValves`. In it, I am going to create an array of `udtValve`. Within this data block, I can set configuration values, including valve names, for each of the valves I am going to use. Let’s imagine we’ll be using 10 valves. As you can see below, I’ve entered configuration values in each valve’s data structure.
Siemens FB for udtValve

The final step on our PLC is to add our newly written code. I've created a FB (*fbValves*) to contain all of my valves, instantiated 10 valves (multi-instantiated versions of *fbValve*) within it, and dropped *fBValves* into OB1.
Now, let's move to the HMI. Let's say we want to put together a simple P&ID of our system showing each of the valves and their status. Let's also imagine we want to allow a user to click on a valve and open a pop-up so that they can get more information as well as control it manually.

Let's start with the first task - our simple P&ID. I'm going to grab a simple valve from the toolbox (under "Elements - Symbol Library") and add an indicator for the valve name. Under "Properties - Appearance" for the symbol, I'm going to change the "Fill Style" to "Shaded". This will allow me to add some color animation to the valve: Orange/flashing when the valve state is not known, Grey when the valve is closed, and Green when the valve is open. Additionally, I would like the valve's name to change to Red and flash when the valve is faulted. I could start adding in my animations now, but I don't want to have to redo that process for each valve. Instead, I'm going to create a faceplate so I can write the logic once and reuse it several times.
times. To do this, select both the valve symbol and the valve name indicator, right-click, choose "Create Faceplate", and name it 'ValveIndicator'.

Inside the faceplate editor, I could create new properties for the fault status, the valve state, and the valve name, then tie each to the appropriate item/animation. However, this would require linking each of these properties back to the appropriate tag inside of my data structure. With a lot of valves, this could take a lot of work. This is where one of the new TIA Portal V13 SP1 updates will come in handy: the ability to use PLC UDTs (PLC data types) in the HMI. To have access to a PLC data type on the HMI, we will first need to add it to our project library as a type. To do so, just drag udtValve from the PLC project tree into the "Project Library - Types" folder:
Now, back to our Faceplate. Let's add a new property of type `udtValve`. Now when we use this Faceplate, we will only need to link a single tag: all of the individual animations and properties will be internally linked, within the Faceplate, to the appropriate tag within our UDT. Below is my Faceplate and the animation for the valve color. Notice in the Properties window that there is only a single property of type `udtValve`.

![Image of Faceplate and animation](image)

When you're finished editing your Faceplate, release the version and let's go back to editing our HMI screen. Select the new Faceplate object and view the Interface tab. Link it back to `dbValves.Valve[0]` by navigating to "Program Blocks - dbValves".

That's it! Now we can add more instances of our Faceplate to the screen and simply link a new valve instance to each - nothing to it!
So, getting back to our original goal - we've succeeded in creating a simple Faceplate that can be linked directly to a single tag (UDT).

The second task we set out to accomplish was to create a pop-up that can show more detailed information for any given valve. In this case, we want to create our pop-up so that it is capable of being opened to display any one valve at a time. To do this, we will need to set up a 'multiplexing' tag to look at different valve instances in our array of udtValve. If you've tried this before, you may have run into issues. Luckily, there have been a few updates in V13 SP1!

First, let's set up our 'multiplexing' tag. We'll need to create an 'index' tag locally on the HMI that can be used to choose the active valve. In my default tag table, let's first create a local Index tag (with type of "UInt"). Next, in your project tree and while still looking at your HMI tag table, select dbValves. From the "Details View" (see screenshot below), drag Valve[1] over to your HMI's default tag table and change the name to IndexedValve:
At this point, notice that the HMI Tag is linked to a specific valve within our array of *udtValves, dbValves.Valve[1]*. Now, select your new *IndexedValve* tag and open up the property pane below. Select the empty drop-down box for “Address” and choose “HMI Tag”. Now, navigate to your HMI tags and select "Index," the internal tag we created a few moments ago. At this point you will notice that the PLC Tag name shows "<Multiplex Tag>" (instead of *dbValves.Valve[1]*), and the 'address' now shows you a dynamic link to array. By setting our *Index* tag, we can now vary where our *IndexedValve* is pointed.
The last piece of our puzzle is the pop-up. This is another great addition to TIA Portal V13 SP1 (for more info, see this). To create a pop-up, navigate in the project tree under your HMI to

Great! Now, let's create another faceplate to show whatever information you would like. I've created the faceplate at left and configured a single "udtValve" property to link it. It will give the user the ability to switch control modes, open and close the valve, and see status information.
"Screen Management/Pop-up Screens". Add a new pop-up screen and add your new faceplate to it. Select your faceplate and view the "Interface" tab under Properties. Now, link your IndexedValve to the faceplate.

Finally - our last step. Add an invisible button and place it over one of the valves on your P&ID screen. You can do this by using a button from your HMI toolbox and configuring it to be 'Invisible' from within the properties. On the button's Click Event, add the following:
Now, when a user clicks on a valve from the P&ID, the triggered event will first update the index of the valve that is chosen, and then show the pop-up screen.

There you have it! We’ve taken advantage of several really great features that were added in TIA Portal V13 SP1: the ability to use PLC data types (UDTs) on an HMI, the ability to link a faceplate to a single UDT, the ability to multiplex an array of UDTs, and the ability to add a simple pop-up. As I'm sure you can imagine, there are many potential uses for these new features and this example is just one possible application. Good luck programming and let me know if there are any new features you've found that can help me be more efficient in the future!

**User Defined Data Types (UDTs) with A-B**

The Controller Organizer has a folder called Data Types > User-Defined with all the UDTs in the project.

I am of the opinion that every PLC program should rely heavily on UDTs to improve readability, and if you are OOP adept, it can be a great help to organize your classes.

Let’s go over the fundamentals of OOP for a little bit:

- Classes: Classes define the abstract characteristics and behavior of an object. For example, a simple VALVE class would have the characteristics (or attributes) that it can be open or closed (the things it can be), and as far as behavior goes, it could have the methods to open and to close (the things it can do).
- Objects: An object is an instance (occurrence) of a class. In our example, there could be a Valve_001 and a Valve_002, which are both instances of the class VALVE, with the same attributes and methods.

If you look at a valve as an object in a typical industrial automation environment, you should note the following:
• It has inputs and outputs that are specific for the object (proximity switches and solenoids).
• It can be either open or closed
• You can tell it to go open or close.
• It could have an alarm timer, that would tell us if the valve did not open or close in a given time period after a command.
• It might have interlocks, which allow the valve to open or close under certain conditions.

A UDT for this class would fit all these properties and methods in one simple type. But, as always, we can expect further complications of the class VALVE during the realization of a project. To be as flexible as possible, I highly recommend the practice of nesting UDTs, which will become clear along the way.

Let's start with defining our class, and keep in mind that it will have to be easily accessible for maintenance people or other programmers.

If we start at the I/O end, the best method is to create sub-classes called VALVE_IN and VALVE_OUT, which will contain our I/O.

The following example uses RSLogix5000 V16. First, create the sub-classes. From the File menu select New Component > Tag. The following dialog box appears to create and edit the members of the UDT.
Now, make a UDT called VALVE, and nest these sub-UDTs in it:

As you see, I am allowed to take the types I just created as the data type in this UDT. The real advantage of this feature will become clear if you create an object called Valve001 of the type VALVE, and look at the object in the monitor tags window:
Wow! Just by creating a new tag of the type VALVE, it gets all these I/O points right away, and referenced in the program:

Of course, going further with this concept, everything for a valve can be included in one object. Allow me to skip some steps, and show you a possible final result:

The VALVE class is now contained in a UDT called VALVE, which looks like this:
As you see, the class VALVE now consists of the sub-classes VALVE_IN, VALVE_OUT, VALVE_TIMER, VALVE_STATUS, etc.
And an instance of this class, the object Valve001, would look like this:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Style</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valve001</td>
<td></td>
<td></td>
<td>VALVE</td>
</tr>
<tr>
<td>Valve001.In</td>
<td></td>
<td></td>
<td>VALVE_IN</td>
</tr>
<tr>
<td>Valve001.In.Open</td>
<td>0</td>
<td>Decimal</td>
<td>BOOL</td>
</tr>
<tr>
<td>Valve001.In.Closed</td>
<td>0</td>
<td>Decimal</td>
<td>BOOL</td>
</tr>
<tr>
<td>Valve001.Out</td>
<td></td>
<td></td>
<td>VALVE_OUT</td>
</tr>
<tr>
<td>Valve001.Out.Open</td>
<td>0</td>
<td>Decimal</td>
<td>BOOL</td>
</tr>
<tr>
<td>Valve001.Out.Close</td>
<td>0</td>
<td>Decimal</td>
<td>BOOL</td>
</tr>
<tr>
<td>Valve001.Timer</td>
<td></td>
<td></td>
<td>VALVE_TIMER</td>
</tr>
<tr>
<td>Valve001.Timer.OpenTimer</td>
<td></td>
<td></td>
<td>TIMER</td>
</tr>
<tr>
<td>Valve001.Timer.CloseTimer</td>
<td></td>
<td></td>
<td>TIMER</td>
</tr>
<tr>
<td>Valve001.Status</td>
<td></td>
<td></td>
<td>VALVE_STATUS</td>
</tr>
<tr>
<td>Valve001.Status.Open</td>
<td>0</td>
<td>Decimal</td>
<td>BOOL</td>
</tr>
<tr>
<td>Valve001.Status.Close</td>
<td>0</td>
<td>Decimal</td>
<td>BOOL</td>
</tr>
<tr>
<td>Valve001.Int</td>
<td></td>
<td></td>
<td>VALVE_INTERLOCKS</td>
</tr>
<tr>
<td>Valve001.Int.Open_Interlock</td>
<td>0</td>
<td>Decimal</td>
<td>BOOL</td>
</tr>
<tr>
<td>Valve001.Int.Close_Interlock</td>
<td>0</td>
<td>Decimal</td>
<td>BOOL</td>
</tr>
<tr>
<td>Valve001.Cmd</td>
<td></td>
<td></td>
<td>VALVE_COMMANDS</td>
</tr>
<tr>
<td>Valve001.Cmd.Open</td>
<td>0</td>
<td>Decimal</td>
<td>BOOL</td>
</tr>
<tr>
<td>Valve001.Cmd.Close</td>
<td>0</td>
<td>Decimal</td>
<td>BOOL</td>
</tr>
<tr>
<td>Valve001.Alarm</td>
<td></td>
<td></td>
<td>VALVE_ALARMS</td>
</tr>
<tr>
<td>Valve001.Alarm.Open_Timeout</td>
<td>0</td>
<td>Decimal</td>
<td>BOOL</td>
</tr>
<tr>
<td>Valve001.Alarm.Close_Timeout</td>
<td>0</td>
<td>Decimal</td>
<td>BOOL</td>
</tr>
</tbody>
</table>

While adding stuff to my class, I did not have to re-create the object Valve001. RSLogix updated it for me, so all the properties and methods are available in my program.

Now, let’s say you’re working on this project with a couple hundred valves, and the customer decides to go with a different type of valve, that also has an analog input, that tells us the exact position of the valve. All we have to do is modify our VALVE_IN sub-class to add this to every instance of the type VALVE:
Of course, you would still have to write code to tell your program what to do with that information, but that is also the reason why PLC programmers still have a job.

For somebody that is not familiar with your program, it might be confusing to look at all your UDTs. We just made eight UDTs for one simple valve class! But remember, you only have to do this during the design phase. Once you have a solid design for all your classes (and made sure their names are self-explanatory), you will never have to look at your UDT folder again, and creating a new instance will be a breeze.
Leveraging Siemens MultiUser Engineering for TIA Portal

Posted by Gina Brooks-Zak in PLC, Automation, Siemens PLC, HMI and SCADA

With Siemens TIA Portal V14, a great new tool for PLC and HMI development called MultiUser Engineering was released.

MultiUser Engineering allows multiple developers to access a server project through local sessions and quickly and seamlessly merge updates to PLC or HMI code such as function blocks, user-defined types (UDTs), WinCC Comfort or Advanced screens, and more. DMC has been using the MultiUser tools with great success, and I'd like to share a few insights with you here.

How Does MultiUser Engineering Work?
MultiUser Engineering stores an alternate version of the typical TIA Portal .ap14 project files to a central location where multiple developers can access the files.

This is called the server project.

For machine builders or large facilities with many TIA Portal projects, it makes sense to create a secured LAN and/or VPN accessible file location for your developers. DMC has set up something similar, but we've also used the MultiUser server in a static IP LAN environment for short commissioning jobs where there is too much overhead to hook up to a wider network. One commissioning engineer can easily set their laptop to host the server and other commissioning team engineers can then access the files by inputting the static IP address of the host laptop. This is a great time saver when hardware configuration or other fundamental project changes common at commissioning time would require all engineers to get a new copy of the project once the updates were downloaded to the PLC.

If you do use this strategy, make sure you always consider network security and use caution, especially if there are any connections to an outside network.
Use HTTP in a static IP environment to quickly use a laptop as the server host

Creating Local Sessions
Once the server project location is established, developers can add projects and create local sessions of projects.

A local session contains the individual changes of a developer. With local sessions, the following can occur:

- Each function block or HMI screen can be flagged by the developer to indicate to everyone else who has a local session open that they are working on that block.
- When the developer is complete with changes, they can be "checked in" to the server project. The other developers can then push those changes to their local sessions by refreshing from the server project.
- If there is any question of who may be working on a certain flagged block, users can right-click on the flagged item and select Usage Info to see the user’s local session and name.
Local instance toolbar at top, check-in, refresh, open server copy, server project status; left, fbPump flagged in local session, displays as blue; right, fbPump displays as yellow in another user’s local session
Top, local session check-in dialogue; bottom, local session refresh from server project view

Right-click on a flag-marked item to view “Usage Info”

Local Sessions versus Online PLC Code

When working online with a PLC using MultiUser Engineering, remember that the server project is not necessarily consistent with the code that is on the PLC. Code changes still need
to be downloaded to the PLC. One developer’s changes do not automatically appear on another developer’s software instance.

That being said, there are many instances where this functionality is advantageous; it allows greater control of the timing of changes and when test code is set to be downloaded to the PLC. An engineer may want to test code online with a PLC before checking in to the server copy while another engineer tests their local session on a completely different PLC.

I have found that in a commissioning environment where you are working on one PLC, it is best to **keep the server project the “consistent” online copy of the PLC**. When one user checks-in, it is their responsibility to refresh their local session and then download the changes to the PLC.

*Remember that the server project copy, local session, and the project online with the PLC may all have different code or pending changes*
Keep in mind that certain changes cannot be made in a local session copy.

These types of changes include:
- **Hardware configuration**
- **PLC or HMI properties**
- **HMI connections**
- **Technology objects**

If the change attempted requires it made directly in the server project, you will receive a prompt at the top of the window, asking to open the server project view.

This task is a bit confusing because once you open the server project, you must then navigate back to what you were trying to add through the project tree on the left-hand side of the Portal UI.

It's important to note that when you or another user is editing the server view, a yellow icon with a lock symbol displays in the local instance toolbar and **no other users can check in their changes until the server view is closed** through clicking Save Changes, Discard Changes or the server symbol in the local instance toolbar.

**Warning dialogue when attempting change in local session which must be made in the server project**

*When opening a project in server view, make sure to navigate to the change you were trying to make. You’ll see both the local session and the server view in your left-hand Portal Project Tree.*

*Users can not check-in while the server project is being edited. Users see a yellow icon with a lock in this case.*

If you are managing the server project file location, make sure to increase the number of stored project versions through the **MultiUser Configuration application** (the default is 3).

If a mistake is made on a check-in, the server project version can be rolled back using the MultiUser Administration application. In addition, I’ve seen a few times where the server project gets locked, and I’ve had to go into the Administration application to unlock or delete a local session.

This tool is very useful for managing all your projects. You will also want to occasionally export and archive a local session consistent with the server project for an extra layer of backup since
the server projects or local session project types cannot be archived like a normal TIA Portal project; the export can be made through TIA Portal in your local session project list.