Can your software engineer program your PLC?

Alastair J. Borrowman*, Philip Taylor.
Observatory Sciences Ltd, William James House, Cowley Road, Cambridge, CB4 0WX, UK.

ABSTRACT

The use of Programmable Logic Controllers (PLCs) in the control of large physics experiments is ubiquitous[1, 2, 3]. The programming of these controllers is normally the domain of engineers with a background in electronics, this paper introduces PLC program development from the software engineer's perspective.

PLC programs provide the link between control software running on PC architecture systems and physical hardware controlled and monitored by digital and analog signals. The higher-level software running on the PC is typically responsible for accepting operator input and from this deciding when and how hardware connected to the PLC is controlled. The PLC accepts demands from the PC, considers the current state of its connected hardware and if correct to do so (based upon interlocks or other constraints) adjusts its hardware output signals appropriately for the PC's demands. A published ICD (Interface Control Document) defines the PLC memory locations available to be written and read by the PC to control and monitor the hardware. Historically the method of programming PLCs has been ladder diagrams that closely resemble circuit diagrams, however, PLC manufacturers nowadays also provide, and promote, the use of higher-level programming languages[4].

Based on techniques used in the development of high-level PC software to control PLCs for multiple telescopes, this paper examines the development of PLC programs to operate the hardware of a medical cyclotron beamline controlled from a PC using the Experimental Physics and Industrial Control System (EPICS), which is also widely used in telescope control[5, 6, 7]. The PLC used is the new generation Siemens S7-1200 programmed using Siemens Pascal based Structured Control Language (SCL), which is their implementation of Structured Text (ST). The approach described is that from a software engineer's perspective, utilising Siemens Totally Integrated Automation (TIA) Portal integrated development environment (IDE) to create modular PLC programs based upon reusable functions capable of being unit tested without the PLC connected to hardware. Emphasis has been placed on designing an interface between EPICS and SCL that enforces correct operation of hardware through stringent separation of PC accessible PLC memory and hardware I/O addresses used only by the PLC. The paper also introduces the method used to automate the creation, from the same source document, the PLC memory structure (tag) definitions (defining memory used to access hardware I/O and that accessed by the PC) and creation of the PC program data structures (EPICS database records) used to access the permitted PLC addresses.

From direct experience this paper demonstrates the advantages of PLC program development being shared between electronic and software engineers, to enable use of the most appropriate processes from both the perspective of the hardware and the higher-level software used to control it.

Keywords: EPICS, PLC, Siemens, Software Engineering, Structured Control Language (SCL), Totally Integrated Automation (TIA) Portal.

1. INTRODUCTION

In an experiment requiring control of a telescope or synchrotron a PLC often provides the link between user interactions on a PC and hardware controlled by electronic signals. Routinely development of the code running on the PC is carried out by a software engineer and development of code running on the PLC is carried out by an electronics engineer, who also specifies the PLC type and electronic connections to the hardware. The separation of responsibilities between disciplines is marked by the communication cable connecting the PC and PLC. On the cable is carried data as defined in the ICD. The ICD is primarily developed by the electronics engineer with the acceptance of modifications from the software engineer to enable correct hardware control from the PC and to ease or improve the PC's code development.

*ajb@observatorysciences.co.uk; phone +44 (0)1223 508259; www.observatorysciences.co.uk

The development responsibilities of the control system for the Bhabha Atomic Research Centre (BARC) and Indira Gandhi Centre for Atomic Research (IGCAR) 15kW medical cyclotron at the Variable Energy Cyclotron Center (VECC) [8], Kolkata, India, have not been separated following this respected pattern. For this project responsibility for PLC code development and ICD have been assigned to the software engineer, while specification of the PLC and hardware connections remain the responsibility of the electronics engineer.

This delineation of responsibilities gave the software engineer flexibility on deciding whether project requirements for control of the hardware were fulfilled in the PC's program design or the PLC's. As PLC programming support onsite will be more difficult than for the PC it was decided that the PLC's processing of demands from the PC would be kept to the minimum required for safe operation of the hardware and that the PLC to PC ICD would include detailed status information to be read by the PC to determine the PLC's process reasoning, e.g. the reason why the PLC is currently not able to move hardware to position demanded by the PC.

The PLC selected was the Siemens S7-1200 that includes some on-board I/O and the ability to expand this by plugging in I/O modules suitable for controlling the connected hardware. The configuration of the PLC, its connected I/O modules and source code development are done using the Siemens STEP 7 Totally Integrated Automation (TIA) Portal IDE running in a Windows virtual machine on a Linux host. The same virtual machine is used for ICD creation using Microsoft Office Excel 2007. Development of the PC EPICS software controlling the PLC is done on the Linux host but can also be built and run on Windows through use of Cygwin.

This paper's findings are not dependent upon use of this specific PLC, its programming software or the use of EPICS on the controlling PC. Other PLC manufactures, e.g. Allen-Bradley, offer PLCs of similar specification and supply similar tools for their configuration and programming. Likewise other toolsets and infrastructures enable development of high-level PC based programs for PLC control, e.g. the Daniel K Inouye Solar Telescope (DKIST), formerly known as the Advanced Technology Solar Telescope (ATST), Common Services Framework (CSF)[3]. It is the authors' assertion that the processes outlined in this paper are equally valid irrespective of PLC selected and programming tools used, however, to fully describe these processes as carried out by a software engineer this paper does make specific reference to their functionality and provided working environment.

2. FROM HARDWARE SPECIFICATION TO ICD

The project's electronic engineer specified the PLC and its connected I/O modules based upon the hardware to be controlled. In the medical cyclotron beamline project the hardware consists of vacuum valves, pumps and gauges, horizontal and vertical slits and beam diagnostic equipment (Faraday cups, beam position monitors and viewing screens), which require a combination of digital and analog input and output signals to control. The information detailing the physical connections between hardware and PLC I/O was provided to the software engineer in the *signal list*. This document consists of a number of tables, each table listing all the connections made between a single PLC I/O module and the hardware connected to it. Importantly it details for each I/O module's port (or channel) what the port is connected to, e.g. for an 8 port digital output module the table consists of 8 rows, each row describing the hardware wired to the port and the relationship between the hardware and a digital output signal of high (1/true) or low (0/false). Ports not currently used are noted as such. A portion of the signal list table detailing a digital output module's connections is shown in Table 1.

Table 1 Section of table from signal list document describing output module connections

I/O Module Reference	Port	Signal Label	Signal Type	Description
K1	DQ a.0	C4VSV2-A	Digital out	Beamline 4 common vacuum gate valve #2, value of 1 opens the valve
K1	DQ a.1	H4VSV1-A	Digital out	Beamline 4 high energy vacuum gate valve #1, value of 1 opens the valve
K1	DQ a.2	L4VSV1-A	Digital out	Beamline 4 low energy vacuum gate valve #1, value of 1 opens the valve

The electronics engineer also detailed the order in which the I/O modules are connected to the PLC in the *controls rack layout diagram*. The order modules are connected to the PLC determines the address used by the PLC program to access

each signal, e.g. the signal C4VSV2-A (beamline 4 common vacuum gate valve #2, see Table 1) is assigned to the first output port (DQ a.0) of the module referenced as K1 (I/O mounted onboard the PLC) and is addressed as Q0.0 in the PLC code. Siemens use the prefix I to address an input signal and Q for output.

Using the supplied signal list and rack layout documents together with the software specification, detailing the overall requirements of the control system as provided by the combined PC and PLC programs, the software engineer began developing the PLC to PC ICD.

To ensure correct specification of the connected I/O addresses in the PLC the Siemens TIA Portal was used to define the hardware in use. This enables specification (by part number) of the PLC and connected I/O modules to transfer the rack layout diagram information into a representation used to configure the PLC and its I/O. This is shown in Figure 1, with the PLC in slot 1 and slots 2 to 9 containing the specified I/O modules in their specified order.

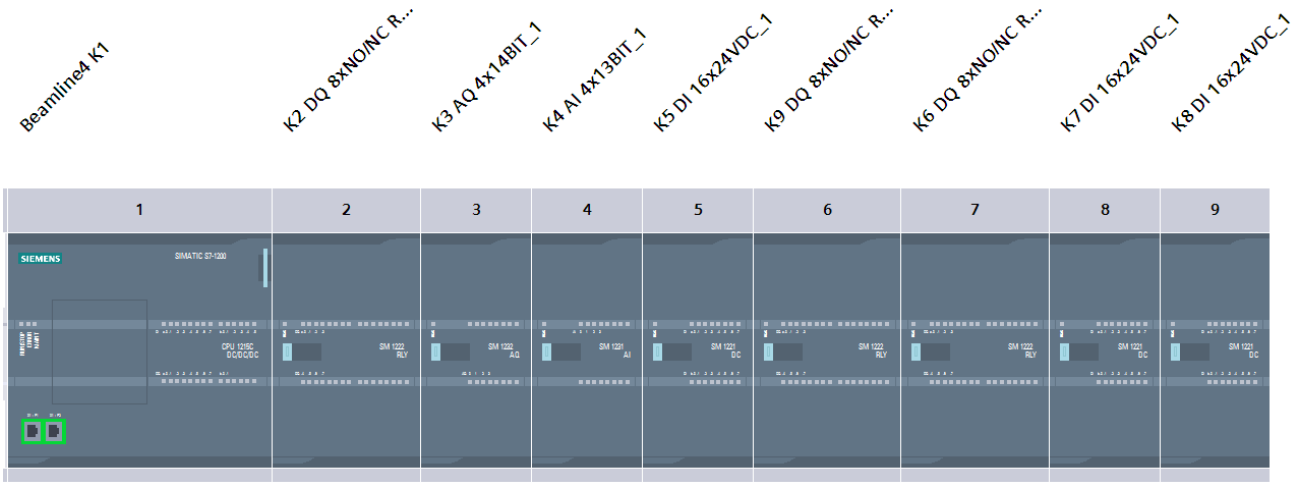

Figure 1 Example of PLC and I/O configuration as displayed in TIA Portal

In addition to I/O memory addresses the PLC uses PLC *bit memory* addresses to accept input from the PC and store status that can be read by the PC. Siemens uses the prefix M to address bit memory.

I/O and bit memory addresses are accessed in the PLC program by their associated symbolic or *tag* name. If an address is not given a tag name a default one is supplied by TIA Portal. To ease navigation of tags they can be grouped arbitrarily into tables, TIA portal places a tag not allocated a table into the default table. Therefore to fully define a PLC tag the ICD must include the following information:

- Tag name — this project incorporated the *Signal Label* from the signal list (see Table 1) into defined tag names to ensure easy traceability of signal from electronics diagram to PLC to PC.
- PLC tag table — this project grouped tags relating to specific I/O modules and operations into named tables.
- Data type — types available depend upon the PLC but commonly used are Boolean (1 bit), byte (8 bits), integer (16 bits), double integer (32 bits), etc. The Siemens S7-1200 also provides the ability to implement *user-defined* types, this enables the same PLC data type to be used in the declaration of multiple PLC tags that contain the same data in the same I/0 address order[9]. This is useful in the development of reusable PLC program functions capable of controlling similar hardware (see section *4.2 Code reuse* of this paper).
- Logical address — the I, Q or M address referenced by this tag name.
- Comment — description of tag and how it is used.

TIA Portal can import into a PLC program its tag definitions from a correctly formatted Excel spreadsheet containing all of the above information. Defining and modifying the tag definitions is less error prone and time consuming using a spreadsheet and import functionality rather than within TIA Portal directly. For this reason (and others explained in section 2.1) it was decided to define all PLC tags used by the PLC in the ICD, not only those directly used to transfer information between PLC and PC.

Figure 2 shows a section of the PLC to PC ICD spreadsheet data that is imported into TIA Portal to define tags controlling the three vacuum gate valves belonging to the project's *beamline 4*. The first row of data defines the tag *C4VSV2_valve* that is used to operate the valve defined in the first row of the signal list shown in Table 1.

	A	B	C	D	E	L	Q
1	Tag Name	PLC Tag Table	Data Type	Addres	PLC Tag Comment	EPICS Record Name	OSL Test PLC Address
26	C4VSV2_valve	K1TagTable	Bool	%Q0.0	Control of beamline 4 common vacuum gate valve 2: 1 open; 0 close		%M1600.0
27	EPICS_C4VSV2_valve	K1EpicsTagTable	Bool	%M1000.0	EPICS input to PLC requesting beamline 4 common vacuum gate valve 1 is opened/closed: 1 open; 0 close	C4:$(PLC):VSV2:OPEN	%M1000.0
28	EPICSPI_C4VSV2_valve	K1EpicsTagTable	Bool	%M1200.0	EPICS PI value of input to PLC requesting beamline 4 common vacuum gate valve 1 is opened/closed: 1 open; 0 close		%M1200.0
36	H4VSV1_valve	K1TagTable	Bool	%Q0.1	Control of beamline 4 high energy vacuum gate valve 1: 1 open; 0 close		%M1600.1
37	EPICS_H4VSV1_valve	K1EpicsTagTable	Bool	%M1000.1	EPICS input to PLC requesting beamline 4 high energy vacuum gate valve 1 is opened/closed: 1 open; 0 close	H4:$(PLC):VSV1:OPEN	%M1000.1
38	EPICSPI_H4VSV1_valve	K1EpicsTagTable	Bool	%M1200.1	EPICS PI value of input to PLC requesting high energy vacuum gate valve 1 is opened/closed: 1 open; 0 close		%M1200.1
46	L4VSV1_valve	K1TagTable	Bool	%Q0.2	Control of beamline 4 low energy vacuum gate valve 1: 1 open; 0 close		%M1600.2
47	EPICS_L4VSV1_valve	K1EpicsTagTable	Bool	%M1000.2	EPICS input to PLC requesting beamline 4 low energy vacuum gate valve 1 is opened/closed: 1 open; 0 close	L4:$(PLC):VSV1:OPEN	%M1000.2
48	EPICSPI_L4VSV1_valve	K1EpicsTagTable	Bool	%M1200.2	EPICS PI value of input to PLC requesting low energy vacuum gate valve 1 is opened/closed: 1 open; 0 close		%M1200.2
297	gateValveSwitches	CompositeTagTable	"type6VacValveSwitches"	%I0.0	Array containing status of 6 vacuum gate valves' open and closed switches status; index 1 is valve 1 open switch status, index 2 is valve 1 closed switch status, index 3 is valve 2 open switch status, etc.		%M1400.0

Figure 2 Section of PLC to PC ICD showing data imported into TIA Portal to define PLC tags

To create the PLC tags only data in columns A to E of the spreadsheet are imported into TIA Portal, the purpose of columns L and Q are described in sections 3 and 5 of this paper respectively.

2.1 PLC scan cycle, I/O updates and PC access to PLC addresses

The processor of the PLC runs the developed *user program* (downloaded to the PLC using TIA Portal) as part of its *scan cycle*. The default maximum processing time for each scan cycle on the Siemens S7-1200 PLC is 150ms, though our project's scan cycle completes in tens of milliseconds. In addition to running the user program during each scan cycle, the PLC also writes to its outputs (Q addresses), reads its inputs (I addresses) and performs housekeeping. During user program execution the values of the Q and I addresses are not accessed directly from the physical I/O but from the PLC's *process image*. Using the TIA Portal default configuration the address values in the process image are updated at a specific time in the PLC's scan cycle:

1. At the start of a scan cycle the output address values contained in the process image are written to the physical outputs. At PLC power-up the process image will contain default (or previously stored) values for all outputs, once the PLC is running the values written will be the final values set during the last processing of the user program.
2. Once output addresses are written the input addresses are read from the physical inputs into the process image. This ensures that no changes to the physical inputs are seen during user program processing, which gives a consistent view of the hardware throughout this scan cycle's processing of the user program.
3. Following write of the outputs from the process image and read of the inputs into the process image, the PLC executes the user program which updates the process image's output addresses as required to operate the hardware correctly as demanded by the PC. As the programmer explicitly defines the order of processing in the user program, the engineer therefore ensures the final value written to the output is correct and due to the PLC's use of a process image, this is the only value written to the physical output; no temporary setting of output values during user program processing are transferred to the hardware.

The Siemens PLC offers other I/O update processing options, as it is assumed other PLC manufacturers do, it also provides programming instructions that give immediate access to physical hardware values, however for our project no other I/O update pattern or direct access has been required to control the hardware or to fulfil project requirements, e.g. as regards response and update times.

To ensure PC hardware demands never by-pass the PLC processing responsible for safe movement of the hardware, the PC program (EPICS Input/Output Controller (IOC) in this project's case) never writes directly to the PLC output hardware addresses but **only** to PLC bit memory (M) addresses specified in the ICD as accepting EPICS demands. The tag names of these addresses are prefixed by *EPICS_* in the ICD and their M address have a value calculated from the Q address of the hardware the demand relates to. E.g. the tag *C4VSV2_valve* obtains its PC demand from tag *EPICS_C4VSV2_valve* (see second row of data in Figure 2), which uses the address M1000.0, i.e. the M address that is the Q address +1000. To ensure the M addresses of the EPICS_ tags are correctly defined (and automatically updated when Q addresses are changed) they are programmatically calculated using a Visual Basic for Applications (VBA) user defined function (UDF) that accepts as input their associated hardware output tag's Q address.

The PLC's use of a process image to ensure consistency of physical I/O values throughout a scan cycle only relates to addresses specified as Q or I hardware addresses. The PLC's bit memory addresses have no relationship with the process image. When an M address is written either by the PLC's user program or by a program running on the PC, all subsequent program instructions in the current scan cycle will read the new value. This provides the opportunity for the PC demand value to change during the processing of checks to determine whether the demand is valid, which may result in the calculated result being wrong. To ensure this does not occur the software engineer designed the PLC user program to include creation of an *EPICS process image*. The ICD uses the prefix *EPICSPI_* to identify addresses that store the EPICS process image values, the address value is once again calculated based upon the Q address of the hardware the demand relates to using a VBA function.

The first user program's PLC organisation block (OB) run on each PLC scan cycle creates the EPICS process image, all subsequent OBs access the PC's demands using their EPICS process image tags. E.g. the PC demand written to tag *EPICS_C4VSV2_valve* is copied to *EPICSPI_C4VSV2_valve* (see third row of data in Figure 2), which is then used by the OB responsible for valve movement when setting value of *C4VSV2_valve*.

Figure 3 shows the PLC processing carried out during each scan cycle when the PLC is running, including the creation of the EPICS process image by the user program.

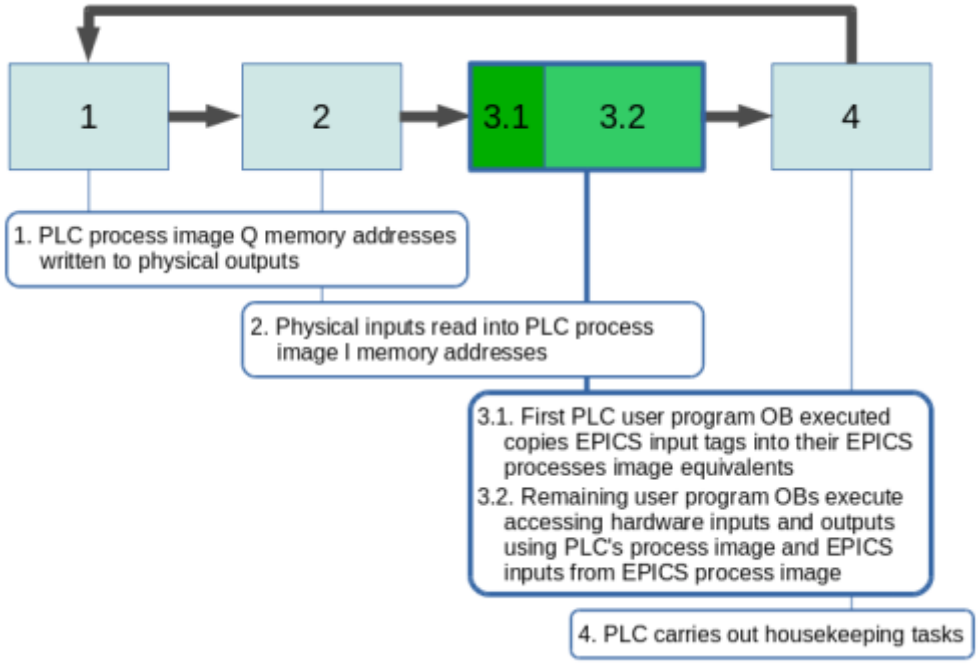

Figure 3 PLC scan cycle processing steps with user program execution highlighted

3. AUTOMATING PC DATA STRUCTURE CREATION FROM ICD

The ICD defines the PLC addresses to be used by the PC control program (EPICS IOC in this project) to send demands to the PLC to move its connected hardware and read status. Irrespective of programming framework used to develop and run the PC program the address and/or tag information from the ICD will require to be represented in a suitable data structure appropriate for the selected framework. Failure to transfer this information correctly from ICD to the relevant data structure will result in failure of the control system, moreover it is highly likely that during project development and deployed lifetime, addresses and tag names will be changed resulting in the data structure definitions requiring corresponding changes. In the medical cyclotron beamline project the software engineer has used the ICD document as the single definition of all PLC addresses and as the single source used to automatically generate the PLC tag definitions and PC program's PLC access data structure definitions.

The automatic generation of PLC tags from the ICD is accomplished by creating the document as a Microsoft Excel spreadsheet and using the TIA Portal's tag import feature, as described in the previous section.

In an EPICS based PC control program (IOC) the data structures used to access the PLC addresses are EPICS database records, other frameworks have other constructs for representing this kind of data, e.g. *properties* in the DKIST CSF[10]. Irrespective of framework used the automatic generation of data structure definition requires the same process; extraction of the relevant data from the ICD and creation of source files in the correct format for use by the framework.

An EPICS IOC program loads its database records from a .db file[11]. The database file entry defining the record instance used to control beamline 4's common vacuum gate value #2 is:

```
record(bo, "C4:PLC1:VSV2:OPEN") {
    field(DESC, "Open/close C4 VSV2 vacuum valve")
    field(DTYP, "s7nodave")
    field(OUT, "@C4_PLC M1000.0")
    field(SIML, "C4:PLC1:SIM:ON")
    field(ONAM, "OPEN")
    field(ZNAM, "CLOSE")
}
```

Code 1 EPICS .db file entry defining record instance that writes data to the PLC

The above defines a binary output (`bo`) record with the name `C4:PLC1:VSV2:OPEN`. The field `DTYP` (device type) specifies the device support module (`s7nodave`) to be used to communicate with the hardware specified in the `OUT` (output) field. This module is the selected EPICS library used to carry out all PC/PLC communication. The `OUT` field value consists of two parameters; the first `@C4_PLC` is the name given to the communications port opened to the PLC during the EPICS IOC's initialisation, the second `M1000.0` specifies the PLC address to be written to when the record is processed following user input requesting the valve be moved. Other fields specify the name of the EPICS record used to turn on/off simulation of all records communicating with the PLC and the values of the record when accessed as a string (e.g. for graphical display purposes). It can be seen that with minor changes (name, description and PLC address) a similar record could be used to control other vacuum valves. Using an EPICS database *template* enables a general definition to be created that includes macros for those items of the record that require a specific value for each record instance. The template used to create the above record instance is:

```
record(bo, "$(BL):$(PLC):$(DEVICEID)$(DEVICENUM):OPEN") {
  field(DESC, "Open/close $(BL) $(DEVICEID)$(DEVICENUM) vacuum valve")
  field(DTYP, "s7nodave")
  field(OUT, "@$(PLC_NAME) $(ADDRESS:OPEN)")
  field(SIML, "$(TOP_BL):$(PLC):SIM:ON")
  field(ONAM, "OPEN")
  field(ZNAM, "CLOSE")
}
```

Code 2 EPICS database .template file definition used to create record instance shown in Code 1

Each `$(macro_name)` is substituted for a defined value at record instance creation when the .db file is built. The macros that require substitution based upon information from the PC to PLC ICD are `$(DEVICEID)`, `$(DEVICENUM)` and `$(ADDRESS:OPEN)`. EPICS provides the Macro Substitution and Include Tool (MSI) that

accepts as input a .substitutions file defining values to be used for each template macro[12]. Below is a section of the substitutions file showing the declaration used to instantiate the three beamline 4 vacuum gate value records using the template shown in Code 2:

```
file vacuumValve.template {
  pattern {TOP_BL, BL, DEVICEID, DEVICENUM, ADDRESS:OPEN}
  {"C4", "C4", "VSV", "2", "M1000.0"}
  {"C4", "H4", "VSV", "1", "M1000.1"}
  {"C4", "L4", "VSV", "1", "M1000.2"}
}
```

Code 3 EPICS .substitutions file declaration used to create three records from database template definition shown in Code 2

The `file` keyword declares the database template file to be used, inside the *file block* the `pattern` keyword declares the macros of the template that are to be instantiated, following this are a number of blocks (one for each record to be created) containing a comma-separated list of the macro values for each record in the order defined by the pattern. Note, the template macros `PLC` and `PLC_NAME` are not instantiated in this substitutions file as their values are not dependent upon ICD hardware data.

In order to use the ICD as the source of the database record instances it is the contents of the *substitutions* file that require to be obtained from the ICD. In order to do this the EPICS database build process (that takes as input a .substitutions file, using a number of database .template files, to build a .db file) was supplemented by a Python script responsible for creating the .substitutions file.

The Python script accepts as input a .csv version of the ICD spreadsheet, reads information about available database templates and the macros they require from an XML file and outputs the .substitutions file used to create the EPICS database. Execution of the script is integrated into the EPICS database build process so that PLC address information is always defined from the source (ICD) when the database is built; it is not a separate process. XML was selected to describe the contents of the database templates as the use of XML was a requirement of the client to describe beamline configurations, therefore it seemed inappropriate to introduce another technology for this purpose. The XML used to describe the vacuum value template is:

```
<dbTemplateFile name="vacuumValve.template">
  <description>template to create vacuum valve records</description>
  <device id="VSV"/>
  <pattern id="TOP_BL"/>
  <pattern id="BL"/>
  <pattern id="DEVICEID"/>
  <pattern id="DEVICENUM"/>
  <pattern id="ADDRESS:OPEN"/>
</dbTemplateFile>
```

Code 4 Example XML definition describing contents of EPICS database template file

The Python script parses the XML and contents of the ICD spreadsheet's *Address* and *EPICS Record Name* columns (see Figure 2 columns D and L respectively) to create the substitutions file. The script also accepts an XML file detailing the database templates to be used by the beamline, this enables the same process to be used for the creation of multiple EPICS IOC .db files specific to various beamlines, that have similar but not exactly the same hardware. To produce another beamline's .db all that is required is for its interface details to be added to the ICD and creation of an XML file listing the database templates providing control of the beamline's hardware. Figure 4 shows the build process from ICD source and XML files to IOC .db file.

Figure 4 EPICS IOC database build process with additional Python step to create substitutions file

3.1 Ensuring developed PC code does not write to the wrong PLC memory addresses

As detailed in section *2.1 PLC scan cycle, I/O updates and PC access to PLC addresses* of this paper, to ensure the PLC does not by-pass PLC program checks to ensure hardware movements received from the PC are currently safe to be carried out, the PC never directly writes to PLC output memory (Q) addresses. Moreover to ensure PC demands received by the PLC remain consistent throughout a PLC scan cycle the PLC creates an EPICS process image using PLC bit memory (M) addresses, that also should never be directly written to by the PC. This means the following two rules can be applied to minimise the possibility that the PC is not accessing PLC memory addresses not defined for its use:

1. The PC program shall never access PLC output (Q) addresses.
2. The PC program shall only access PLC bit memory (M) addresses within a specified range.

Compliance to the above rules can be checked in PC EPICS code by executing the appropriate search of the IOC's .db file (e.g. using the Linux command-line grep utility or additions to the build process that give a build error if an incorrect address is used). Other PC frameworks will also enable such checks to be carried out, preferably on a text file in the toolchain's processing just prior to compilation/interpretation.

The above tests check only that the PC program writes to the correct memory type and within the correct range of memory addresses, they do not test whether the PC is writing to the correct address as defined in the ICD when attempting to move a specific item of hardware, e.g. address M1000.0 to move beamline 4's vacuum gate value #2. This test can be implemented by development of an ICD reverse engineering process. The process creates a version of the ICD from the PC program's source, which is then checked for accuracy against the actual ICD. For example, to test a PC EPICS program the input for ICD reverse engineering would be the EPICS IOC's database .db file, from which all ICD entries defining PLC memory addresses used by the PC can be produced. The reverse engineering process outputs a file of suitable format to enable easy clarification that the PC program is correctly using the data from the ICD. Caution must be used when developing such a process to ensure that bugs in the ICD to PC data structure creation are not reproduced and masked in the reverse engineering of the ICD.

4. PLC PROGRAMMING

4.1 Introduction

The Siemens S7-1200 PLC's user program is developed using the TIA Portal IDE. The programming language selected for this project, following guidance in the Siemens Programming Styleguide, was the Structured Control Language (SCL)[4], which is Siemens' implementation of IEC 61131-1 standard Structured Text. This is a Pascal based language with necessary additions for PLC programming, e.g. hardware access, timer and counter creation and monitoring instructions, etc[9]. The language supports the PLC organisation block (OB) program structure used to organise the user

program into a sequential series of independent but related programs run on each PLC scan cycle. The run sequence is determined by the defined OB's number; OB(n) is not executed until OB(n-1) has completed.

4.2 Code reuse

Code reuse is a software engineering axiom and to facilitate this Siemens provides programming support for Functions (FCs) and Function Blocks (FBs), the difference being an FB may declare static data persistent across scan cycles. FCs and FBs may be placed into a *Project library* enabling them to be called by multiple PLC user programs within a single TIA Portal project, or into a *Global library* allowing them to be imported into multiple TIA Portal projects. The medical cyclotron beamline TIA Portal project contains two PLC device configurations (for two different beamlines) and uses a Project library of FCs to enable the same code to be used by both PLC's user programs. Combining the use of a Project library FC and user-defined type (analogous to C/C++ typedef) enables the same function to be used to operate hardware in multiple PLC user programs.

For example, the *GetVacuumValvesOpen* Project library FC accepts as input the user-defined type *type6VacValvesSwitches* and returns true if and only if all vacuum valves are open. The final row of ICD spreadsheet section shown in Figure 2 defines the tag *gateValveSwitches* using this type, with the starting memory address of the tag defined as I0.0. As described in the tag's comment the type declares an array of 6 Booleans used to access the memory addresses of three vacuum valves' open and closed switches; index [0] of a tag using this type will contain the status of the first valve's open switch (as accessed by the starting memory address defined for the tag), index[1] will contain the status of the first valve's closed switch and as the array is of type Boolean this will have address +1 byte from index[0] (i.e. I0.1), index[2] will contain the second valve's open switch status (address I0.2), etc. The ICD of another beamline can use the same type in a tag definition but may define its starting address as I2.0, as this is the address of its first vacuum valve's open switch. Using a Project library containing the FC and user-defined type enables PLC user programs of both beamlines to use the same code irrespective of differences in their I/O modules hardware connections.

To further facilitate code reuse TIA Portal provides the *Variant* data type that is analogous to a C/C++ pointer to void. Using a Variant data type as a Project library FC's input parameter allows the function's use to be expanded to accept different data types at runtime. The actual type of the Variant can be tested at runtime using the *TypeOf* instruction, which combined with a *CASE* statement enables the function to determine the current type and process its data appropriately. This approach was used to further develop the *GetVacuumValvesOpen* Project library FC modifying the accepted input parameter from *type6VacValvesSwitches* to Variant in order that it could also accept the user-defined type *type4VacValvesSwitches* used by beamlines having only 2 (not 3) vacuum valves.

4.3 Software version control

The importance of software version control is another software engineering axiom; at a minimum it must be possible to easily commit and retrieve from secure storage current and previous versions of the project's source files, from which all executables can be built. The creation of a new project in TIA Portal creates a directory of multiple sub-directories, of multiple levels, containing multiple file types, which presumably could be individually committed to a version control system. However, TIA Portal supports archiving a project into a single file. Following testing with Subversion (SVN) it was determined a project archived using TIA Portal, committed and retrieved from SVN, could then be successfully reinstated into TIA Portal using its project retrieve functionality. Therefore this is the version control method used for securing the PLC configuration and source code of the medical cyclotron beamline project.

5. PLC BASED SIMULATION

Within a project's lifecycle, development of its PLC and PC programs are begun prior to delivery, construction or even final design of the project's hardware. For this reason simulation of the hardware is of upmost importance if the PLC and PC programs design and implementation are going to be rigorously inspected and tested, at a time in the project when modifications will be the least costly in time and expense to realise. Moreover, hardware simulation remains important after hardware delivery and into operations as it provides a useful testing tool for determining location of faults and to enable continued system use during maintenance or hardware failure.

Packaged with TIA Portal is S7-PLCSIM that when used together provide a testing framework for PLC user programs. The user program is executed in TIA Portal and connects to S7-PLCSIM that provides the ability to implement simulation of hardware connected to the PLC's I/O, e.g. following the write by the PLC user program of an output signal the simulator can be programmed to update the PLC's input signals to simulate movement of hardware as demanded.

However, this can only be used when running the PLC program in the TIA Portal, not when running the PLC program on the actual PLC. This means S7-PLCSIM cannot be used to test the correct operation of the PLC program when it is being controlled by a PC program. Therefore, if it is required to run the PLC program on the specified PLC (with or without specified I/O modules connected) and for it to simulate reactions to input demands received from the PC program, it is necessary to add simulation to the PLC's user program. This simulation has the following requirements:

1. Simulation shall only occur when required or requested, by default the PLC will not provide simulation of hardware configured for its control.
2. Implementation of simulation in the PLC's user program shall be separated from non-simulation code, enabling easy removal of the simulation code when/if it is no longer required, without necessitating extensive changes to non-simulation code.

These requirements have been met in the medical cyclotron beamline PLC user programs using the following techniques:

- The PLC to PC ICD contains a column defining PLC memory addresses to be used when in simulation (see Figure 2 column Q *OSL Test PLC Address*). All row entries in this column call the user defined VBA function *plcAddressToOslTestAddress* that accepts as an argument the row's PLC address entry (column D of spreadsheet) and if this is an input (I) or output (Q) hardware address it returns a PLC bit memory (M) address using a specified M address simulation range (+1400 for I address or +1600 for Q address), an argument address containing an M address is returned unchanged.
- When hardware simulation by the PLC is required the PLC memory address column of the spreadsheet imported into TIA Portal to create the PLC's tag definitions, is replaced by the column containing the simulation addresses. Multiple versions of the simulation address column can exist, each containing differing levels of simulation, e.g. simulation of all hardware or simulation of only specific mechanisms (those mechanisms not to be simulated reference their PLC address entry in the simulation address column, rather than calling VBA function *plcAddressToOslTestAddress*). The newly updated ICD spreadsheet is used as the source for importing the PLC tags into TIA Portal and for rebuilding the PC EPICS IOC database file.
- A PLC user program OB is created and configured to only execute when the PLC transitions from STOP mode (user program is not executing) to RUN mode (user program is executing). This OB uses the SCL instruction *IO2MOD*, that determines the I/O module of a given I/O address as referenced by a PLC tag. When given a tag not referencing an I/O module address, e.g. a tag using a PLC bit memory (M) address, it returns an error that signals the mechanism using this tag is in simulation. The simulation status of each mechanism is stored in a tag that is allocated in the ICD to the tag table *SimTable*.
- In each PLC user program OB responsible for moving hardware, once the hardware output tag (referencing an M address when in simulation) has been updated to value demanded by the PC, a check is made as to whether this mechanism is in simulation by examining its SimTable tag. If simulation is *on* a suitable FC is called to modify the output's related inputs (referencing M addresses when in simulation) to simulate the necessary hardware movement. FCs are not always used for simulation and for certain tests their simulation is turned off, e.g. to test PLC's and PC's program responses to hardware not moving as requested, etc. The ability to run such tests in this manner is often easier then modifying the real hardware to perform all failure scenarios in order to test a program responds correctly.
- To remove simulation from the PLC the tag table SimTable is deleted from the PLC's configuration and the PLC user program re-compiled. This will result in errors highlighting the OBs referencing the deleted tags, these OBs are modified (lines commented-out or deleted) to remove all references to the simulation status tags. This will include the startup OB responsible for determining simulation status, which can be fully commented out or deleted as necessary.
- Simulation capabilities can be re-instated if required by retrieving from SVN a suitable version of the TIA Portal project containing the simulation code.

PC only simulation is also available in the medical cyclotron beamline project through use of the EPICS framework's simulation capabilities. Each EPICS record capable of I/O has a number of fields to configure simulation, this project uses the `SIML` (simulation link) field to turn I/O records simulation on/off (see Code 1). All records responsible for PLC

I/O set this field's value to point to the same EPICS record. When this record is set to turn on simulation all records referencing this record in their `SIML` field will set their simulation status *on* and not output any value to their referenced output address or read input from referenced input address. Based upon the level of simulation required other records can monitor the simulation status of output records and provide simulated hardware behaviour by poking values into the necessary input records.

6. CONCLUSIONS

This paper has endeavoured to show the advantages of project PLC and ICD development responsibilities being shared by engineers with software and electronic engineering expertise. From previous experience of developing PC programs to control PLCs and then direct experience of ICD, PLC and PC development for the same project a number of recommendations can be made:

- Use a single source document (ICD) for defining and auto-generating the PLC tags and PC program data structures accessing those tags.
- Define PLC bit memory (M) address ranges to be used for specific tasks, e.g. to accept input from PC. Ensure adherence to these ranges can be tested in developed PC code.
- Use a PC input *processes image* in the PLC user program to ensure PC input values accepted by the PLC program remain consistent during a scan cycle.
- Use respected software engineering processes in the development of the PLC user program code, e.g. code reuse and software version control.

REFERENCES

[1] Raskin, G. et al, "The Mercator telescope: relevance, status, and future", Proc. SPIE 9145, 914543 (2014)
[2] Kuntschner, H. et al, "ERIS: preliminary design phase overview", Proc. SPIE 9147, 91471U (2014)
[3] Borrowman, A. J. et al, "Software control of the Advanced Technology Solar Telescope enclosure PLC hardware using COTS software", Proc. SPIE 8451, 84510I (2012)
[4] Siemens, [Programming Styleguide for S7-1200/S7-1500], Siemens ID: 81318674 V1.1, (06/2015)
[5] Rambold, W. N. et al, "Upgrade and standardization of real-time software for telescope systems at the Gemini telescopes", Proc. SPIE 9152, 915213 (2014)
[6] Gorges, B. et al, "UKIRT remote operations fail-safe system", Proc. SPIE 8451, 84510Z (2012)
[7] Guzman, J. C. et al, "The Australian SKA Pathfinder (ASKAP) software architecture", Proc. SPIE 7740, 77401J (2010)
[8] Variable Energy Cyclotron Center, < http://www.vecc.gov.in> (4 May 2016)
[9] Siemens, [S7-1200 Programmable Controller, System Manual], Siemens ID: A5E02486680-AJ, (06/2015)
[10] Hubbard, J., et al, "A standard framework for developing instrument controllers for the ATST", Proc. SPIE 8451, 84510O (2012)
[11] Kraimer, M. R. et al [EPICS Application Developer's Guide, EPICS Base Release 3.14.12.5], <http://www.aps.anl.gov/epics/base/R3-14/12-docs/AppDevGuide/> (4 May 2016)
[12] MSI: Macro Substitution and Include Tool, <http://www.aps.anl.gov/epics/extensions/msi/> (4 May 2016)
[13] Parr, E.A., [Programmable Controllers An Engineer's Guide], Newnes, Oxford (2003)