16th Int. Conf. on Accelerator and Large Experimental Control Systems ICALEPCS2017, Barcelona, Spain JACoW Publishing
ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA020

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

MATLAB CONTROL APPLICATIONS EMBEDDED INTO EPICS PROCESS CONTROLLERS (IOC) AND THEIR IMPACT ON FACILITY OPERATIONS AT PAUL SCHERRER INSTITUTE

P. Chevtsov[†], T. Pal, Paul Scherrer Institut, 5232 Villigen PSI, Switzerland
M. Dach, Dach Consulting GmbH, 5200 Brugg, Switzerland

Abstract

An automated tool for converting MATLAB based controls algorithms into C codes executable directly on EPICS process control computers (IOCs), was developed at the Paul Scherrer Institute (PSI). Based on this tool, several high level control applications were embedded into the IOCs, which are directly connected to the control system sensors and actuators. Such embedded applications have significantly reduced the network traffic and, as a result, the controls data handling latency. The paper concentrates on the most important components of the automated tool and some performance results of MATLAB algorithms converted by this tool.

INTRODUCTION

EPICS [1] Channel Access (CA) servers, which are also associated with Input-Output Controllers or IOCs, are well suited for interacting with sensors and actuators engaged in the control system. However, basic EPICS doesn't provide powerful standard options for dealing with sophisticated algorithms, which can be used on the IOC directly to process data obtained from sensors and to control actuators. Some simple available means include the calculation (calc) record with embedded elementary mathematical operations and the pid record for typical proportional-integral-derivative control functions. If more advanced mathematical calculations are required, then the dedicated procedures must be written. It can be either a C code to be implemented as a process function of a subroutine (sub) record or a state notation language (SNL) program.

The CA servers are developed and supported by control system specialists. This ensures that these servers are robust and their data are always available for users. At the same time, any efficient data analysis and data processing algorithms can only be developed by scientists who are experts in the fields of science associated with these data. This clear border of responsibilities between control system developers and users allows one to increase the research efficiency of any scientific organization.

Very powerful data processing and modeling features are provided by MATLAB [2], which is used in research organizations worldwide. In particular, PSI scientists and engineers have a long successful experience with MATLAB applications dealing with control system data. MATLAB is deployed at PSI as an integrated part of the well-established EPICS based data management environment [3]. The access to control system data, which are associated with EPICS records and referenced as channels, is provided by two in-house developed interface packages: Java based ca_matlab [4] and C/C++ based MOCHA/CAFE [5]. The latter, for instance, makes all basic CA functionalities available for MATLAB programs in terms of simple and transparent commands. For instance, to get the **value** from a channel with a specified name, one can use the following command:

value = **caget**('channel name').

Similarly, to set the control parameter represented by some channel name equal to the required value the next command can be used:

caput('channel name', **value**).

MATLAB codes are interpreted and executed line by line, which makes it an ideal tool for application prototyping and testing. The performance of the MATLAB programs versus their C program equivalents is, however, questionable. With all its great features, MATLAB is a good solution for "off-line" data analysis applications where a deterministic time response is not mandatory. However, to be useful for real "on-line" data processing applications, MATLAB, with its simplicity of the code development and maintenance, has to be combined with the C code performance and real-time data management mechanisms. The paper describes a way how to implement such a combination in the context of the EPICS CA server closed loop control.

MATLAB PROGRAM INTRACTION WITH EPICS BASED CONTROL SYSTEM

MATLAB programs at PSI are mostly used in closed loop control applications as illustrated in the Fig. 1.

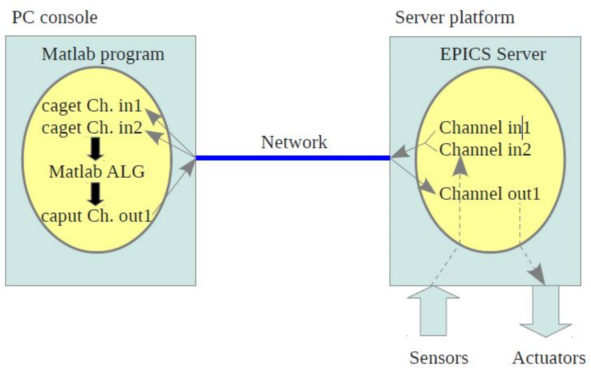


Figure 1: Closed loop controls with the use of an EPICS CA server and a MATLAB program.

† pavel.chevtsov@psi.ch

Data flows and processing are done as following. The EPICS CA server feeds the input (in) channels with data from sensors. The MATLAB program, which runs on a PC console, obtains data from these channels via the computer network and performs all necessary data processing calculations. After calculations, the result data are sent via the network to the appropriate EPICS output (out) channels and further to the corresponding actuators.

This scenario looks fine from the functional point of view. However, the performance of this system depends on several factors. The Ethernet based computer network, which connects clients and servers, is a definitely weak point. By its nature, the network introduces time latencies and jitters while sending data either way, which makes it not time deterministic and, as a result, not optimal for closed loop control applications. Besides, PC consoles housing MATLAB applications and running under Linux or Windows OS also contribute to the overall not time deterministic system behaviour.

A proposal that minimizes unwanted delays for data flows in the above mentioned scenario is shown in Fig. 2. The MATLAB based program is moved from the console PC directly to the IOC. This allows one to simplify interconnections between system components and significantly increase the system performance.

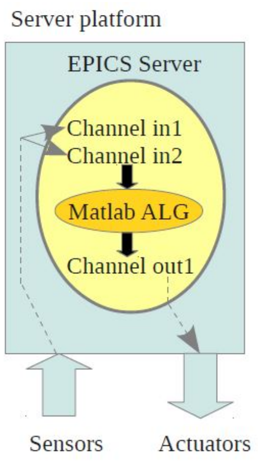


Figure 2: MATLAB algorithms embedded into the IOC.

The basic MATLAB package comes with the MATLAB Coder, which produces a C code from a MATLAB program. According to MathWorks specifications, the resultant C code is functionally equivalent to the original MATLAB program. The original MATLAB programs can be launched on selected computer platforms under Windows, Linux or Mac OS. The embedded systems, which use ARM, Power PC or MIPS processors, cannot run MATLAB codes directly but the C code generated by the MATLAB Coder can be compiled or cross-compiled for all those architectures.

Imagine that the C code is generated from a MATLAB program. How to embed it into the IOC?

One possible way is to make it a part of an EPICS sequencer (SNL) program.

Another way is to use a standard Array Subroutine (aSub) EPICS record. In this case, the input (inp) links of the aSub record are associated with EPICS channels, which are used in the original MATLAB program in the context of caget calls. The output (out) links of this record replace caput calls. The C code generated by the MATLAB Coder is encapsulated by the process function of the record.

To realize these two ideas, an automated tool for converting MATLAB based controls algorithms into C codes was created. The tool is called the MATLAB to C Controls Conversion Tool (MCCCT). Its core consists of three Linux shell scripts.

One script (**coderPreprocessor.sh)** does the MATLAB program pre-processing. It acts on the original MATLAB program and prepares it for the C code generation. The main goal is to replace all **caget** and **caput** calls with the entries required by the aSub record or SNL program (see Fig. 3 in the Appendix). In addition, this script determines the type and number of elements for EPICS channels used by the original MATLAB program.

Two other scripts do the post-processing of the C code generated by the MATLAB Coder.

The first post-processing script (**coderPostprocessor.sh)**, converts the generated code into a suitable aSub record process function. The script also

- generates the EPICS database associated with the aSub record, which has input and output links filled with all channels involved,
- generates a standard IOC startup script to load the aSub record related database, which was generated in the previous step,
- creates a makefile to compile/cross-compile the C code, and
- produces all header files and C files, which are required for the final compilation/cross-compilation.

The second post-processing script (**coderPostprocessor_snl.sh**) inserts the C generated code into a standard SNL skeleton provided by the MCCCT (see Fig. 4 in the Appendix). The script also

- creates a makefile to compile the produced SNL program,
- generates a standard IOC startup script that loads a shared library associated with the resultant
- SNL program, and
- produces a standard IOC SNL startup script that launches the SNL program.

The MCCCT solution looks very attractive for MATLAB/EPICS developments. The original MATLAB code, which is easy to test and maintain, can be used for prototyping, while the generated C code can be used as a production version to be embedded in the dedicated EPICS CA server.

We note that the SNL part of the MCCCT solution is especially efficient for MATLAB programs doing data processing and control actions in infinite loops.

Some more MCCCT details can be found in the Appendix.

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

CONCLUSION

The automated tool for converting MATLAB based controls algorithms into C codes has been in use at PSI for about two years. Based on this tool a number of MATLAB applications dealing with the supervisory of SwissFEL subsystems, such as diagnostics and lasers, were converted and embedded into the IOCs handling sensors and actuators directly. The resultant impact on the machine operations is clearly noticeable, which contributes to overall machine stability.

All control network traffic associated with the communication of such applications with IOCs is eliminated. This is especially important in conditions of SwissFEL operations when the network is heavily loaded not only by a wide variety of control tools used on-line but also by the beam synchronous data acquisition system [6] running at 100 Hz.

After conversion, original MATLAB applications do not have to run on PC consoles anymore. This solves Linux system administrator problems related to the execution of these applications on PC consoles and frees the PC resources for other tasks. Besides, as parts of the IOC software, the converted applications get the around the clock support from the control system team, which ensures their reliable run during SwissFEL operations.

Talking directly to controls hardware, embedded MATLAB applications are much faster than their original versions. An average execution time of control loops was reduced up to 5-10 times. For instance, in case of electro-optical modulator bias scans for SwissFEL bunch arrival time monitors [7] this time was reduced from 75 to 15 seconds.

Further developments of the technique presented in this paper will be concentrated around MATLAB Simulink applications and embedding initial MATLAB algorithms into FPGA modules.

APPENDIX

The Appendix explains in details all the steps, which must be followed to embed a MATLAB code into the IOC with the use of MCCCT.

First of all, a MATLAB program, which interacts with an EPICS CA server, has to be written and tested. After that, one can switch to the conversion procedures. We note that all these procedures were successfully tested at PSI on 64 bit PC consoles running Scientific Linux 6.4.

Assuming that there is a test MATLAB program **myTest8.m** residing in a **/home/ioc/MATLAB-TEST** directory, one switches to this directory

> cd /home/ioc/MATLAB-TEST

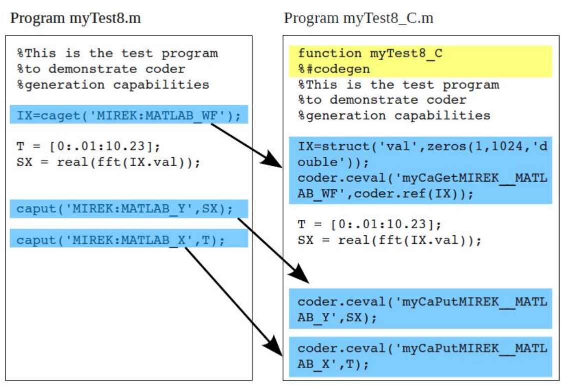
and starts to execute the MATLAB to EPICS conversion pre-processing script by typing

> coderPreprocessing.sh myTest8.m

During the execution, the script reports its actions and results.

```
>Creating myTest8_C.m based on myTest8.m for
 further MATLAB codegen processing
>Inserting function name myTest8_C
>Inserting %#codegen directive
>
>
>
>Examining EPICS server(s) to substitute caget calls
>Examining EPICS server(s) to substitute caput calls
>
>Substituting caget and caput with coder.ceval entries
>1. myTest8.m -> myTest8_C.m                [READY]
```

From the reported results, it is seen that the script parses the myTest8.m MATLAB program and produces an intermediate myTest8_C.m program, which is suitable for the MATLAB Coder use. The difference between these two programs is shown in Fig. 3.


Program myTest8.m

```
%This is the test program
%to demonstrate coder
%generation capabilities

IX=caget('MIREK:MATLAB_WF');

T = [0:.01:10.23];
SX = real(fft(IX.val));

caput('MIREK:MATLAB_Y',SX);

caput('MIREK:MATLAB_X',T);
```

Program myTest8_C.m

```
function myTest8_C
%#codegen
%This is the test program
%to demonstrate coder
%generation capabilities

IX=struct('val',zeros(1,1024,'d
ouble'));
coder.ceval('myCaGetMIREK__MATL
AB_WF',coder.ref(IX));

T = [0:.01:10.23];
SX = real(fft(IX.val));

coder.ceval('myCaPutMIREK__MATL
AB_Y',SX);

coder.ceval('myCaPutMIREK__MATL
AB_X',T);
```

Figure 3: The original myTest8.m program and its pre-processing result.

Meanwhile, the script continues and starts the MATLAB Coder.

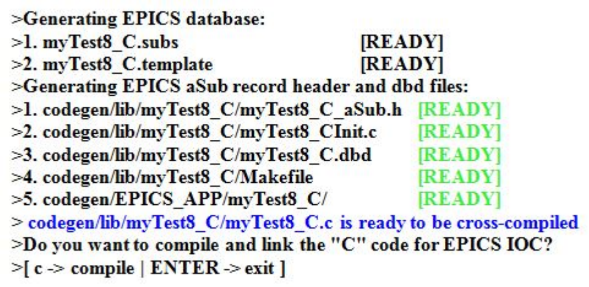
```
>Starting Matlab version 2015a with the command:
>codegen -c -report myTest8_C
>Please be patient, it can take a while...
```

After some work, the C code is produced.

```
>Matlab Coder has generated "C" code myTest8_C.c successfully
> [1] post-process c code for aSub record usage or
> [2] post-process c code to be embed into SNL program
>Type [1 | 2 | ENTER -> exit ] for further processing:
```

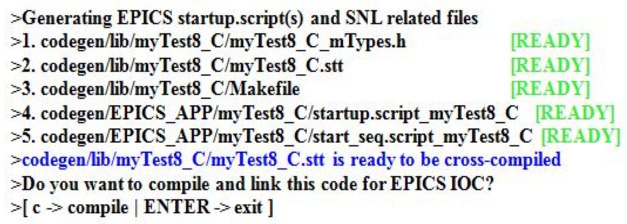
At this moment, it has to be decided if the generated C code is going to be used with the aSub record or to become a part of the SNL program.

When the first option is chosen (“**1**” is typed followed by the ENTER key) then the corresponding post-processing script **coderPostprocess.sh** generates the EPICS database and files supporting the aSub record.



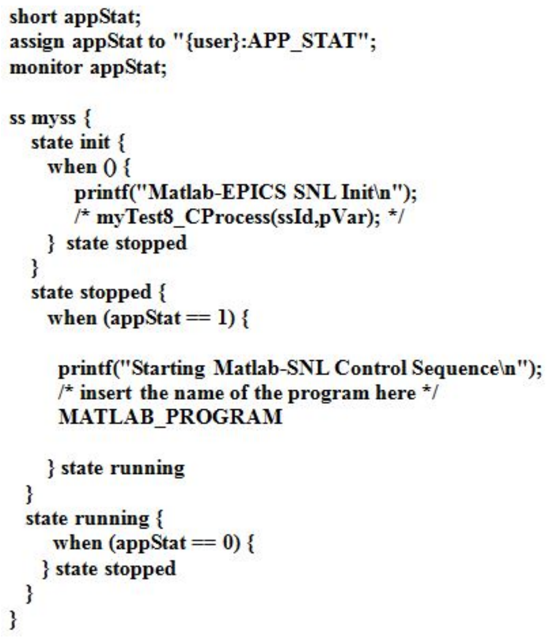
```
>Generating EPICS database:
>1. myTest8_C.subs                          [READY]
>2. myTest8_C.template                      [READY]
>Generating EPICS aSub record header and dbd files:
>1. codegen/lib/myTest8_C/myTest8_C_aSub.h  [READY]
>2. codegen/lib/myTest8_C/myTest8_CInit.c   [READY]
>3. codegen/lib/myTest8_C/myTest8_C.dbd     [READY]
>4. codegen/lib/myTest8_C/Makefile          [READY]
>5. codegen/EPICS_APP/myTest8_C/            [READY]
> codegen/lib/myTest8_C/myTest8_C.c is ready to be cross-compiled
>Do you want to compile and link the "C" code for EPICS IOC?
>[ c -> compile | ENTER -> exit ]
```

In case if one decides to use the second option (“**2**” is typed followed by the ENTER key), the **coderPostprocess_snl.sh** script inserts the generated C code into a standard MCCCT SNL skeleton, which is shown in Fig. 4, and creates files supporting the SNL program.



```
>Generating EPICS startup.script(s) and SNL related files
>1. codegen/lib/myTest8_C/myTest8_C_mTypes.h              [READY]
>2. codegen/lib/myTest8_C/myTest8_C.stt                   [READY]
>3. codegen/lib/myTest8_C/Makefile                        [READY]
>4. codegen/EPICS_APP/myTest8_C/startup.script_myTest8_C   [READY]
>5. codegen/EPICS_APP/myTest8_C/start_seq.script_myTest8_C [READY]
>codegen/lib/myTest8_C/myTest8_C.stt is ready to be cross-compiled
>Do you want to compile and link this code for EPICS IOC?
>[ c -> compile | ENTER -> exit ]
```

After that, in both cases, if “**c**” is typed and followed by the ENTER key, then the compilation/cross-compilation starts. At the end of this process, all necessary shared libraries are created, the corresponding EPICS database support structure (template, substitution and startup.script files) is generated and gets ready for downloading into the IOCs.



```
short appStat;
assign appStat to "{user}:APP_STAT";
monitor appStat;

ss myss {
   state init {
     when () {
        printf("Matlab-EPICS SNL Init\n");
        /* myTest8_CProcess(ssId,pVar); */
     } state stopped
   }
   state stopped {
     when (appStat == 1) {

       printf("Starting Matlab-SNL Control Sequence\n");
       /* insert the name of the program here */
       MATLAB_PROGRAM

     } state running
   }
   state running {
      when (appStat == 0) {
     } state stopped
   }
}
```

Figure 4: MCCCT SNL program skeleton.

REFERENCES

[1] EPICS, http://www.aps.anl.gov/epics/.

[2] B. Hahn and D.T. Valentine, “Essential MATLAB for Engineers and Scientists”, *Academic Press*, USA, 2013, p. 424.

[3] E. Zimoch *et al.*, “SwissFEL Control System. Overview, Status and Lessons Learned”, presented at ICALEPCS'17, Barcelona, Spain, Oct. 2017, paper MOAPL04, this conference.

[4] ca_matlab, https://github.com/channelaccess/ca_matlab

[5] J. Chrin, “An update on CAFE, a C++ channel access client library, and its scripting language extensions”, in *Proc. ICALEPCS'15*, Melbourne, Australia, Oct. 2015, pp. 1013-1016.

[6] S. Ebner *et al.*, “SwissFEL - beam synchronous data acquisition – the first year”, presented at ICALEPCS'17, Barcelona, Spain, Oct. 2017, paper TUCPA06, this conference.

[7] P. Chevtsov *et al.*, “Bunch arrival time monitor control setup for SwissFEL applications”, presented at ICALEPCS'17, Barcelona, Spain, Oct. 2017, paper TUPHA020, this conference.

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.