Programming Manual

DCU Control Design Tool

09/09/14

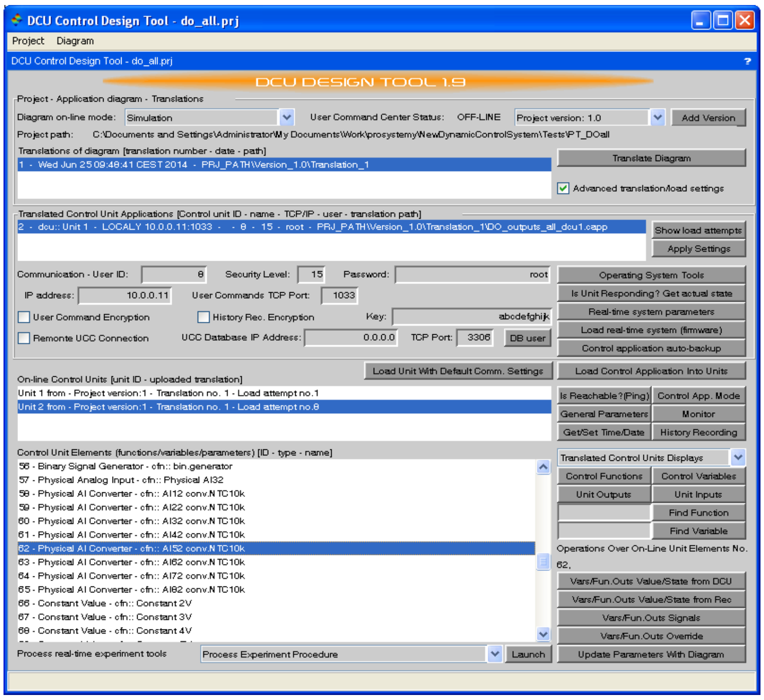
Contents

DCU Control Design Tool toolbox structure

Toolbox overview

Dynamic control unit (DCU) is freely-programmable control unit (controller) with embedded hard real-time system, 16 analog inputs, 8 analog outputs, 32 digital (binary) inputs/outputs, MODBUS RTU connectivity. For more information see [1].

DCU Control Design Tool is an application with graphical user interface that serves for programming, testing, and on-line monitoring of DCU (dynamic control unit) real-time controller. The tool is programmed inside Matlab as well as Scilab scientific computational platforms. Therefore there exists two DCU Control Design Tools, one for Matlab, one for Scilab. The tool is one of many so called modules or toolboxes of these computational platforms. The toolbox for Matlab and the toolbox for Scilab are practically identical. Only minor differences exist due to minor differences in programming languages of Matlab and Scilab.


DCU Control Design Tool - do_all.prj
Project
Diagram
DCU Control Design Tool - do_all.prj
DCU DESIGN TOOL 1.9
Add Version
Translate Diagram
Show load attempts
Apply Settings
Operating System Tools
Is Unit Responding? Get actual state
Real-time system parameters
Load real-time system (firmware)
Control application auto-backup
DB user
Load Unit With Default Comm. Settings
Load Control Application Into Units
Is Reachable?(Ping)
Control App. Mode
General Parameters
Monitor
Get/Set Time/Date
History Recording
Control Functions
Control Variables
Unit Outputs
Unit Inputs
Find Function
Find Variable
Vars/Fun.Outs Value/State from DCU
Vars/Fun.Outs Value/State from Rec
Vars/Fun.Outs Signals
Vars/Fun.Outs Override
Launch
Update Parameters With Diagram

Project - Application diagram - Translations
Diagram on-line mode: Simulation
User Command Center Status: OFF-LINE
Project version: 1.0
Project path: C:\Documents and Settings\Administrator\My Documents\Work\prosystemy\NewDynamicControlSystem\Tests\PT_DOall
Translations of diagram [translation number - date - path]
1 - Wed Jun 25 09:48:41 CEST 2014 - PRJ_PATH\Version_1.0\Translation_1
[✓] Advanced translation/load settings

Translated Control Unit Applications [Control unit ID - name - TCP/IP - user - translation path]
2 - dcu:: Unit 1 - LOCALY 10.0.0.11:1033 - - 8 - 15 - root - PRJ_PATH\Version_1.0\Translation_1\DO_outputs_all_dcu1.capp

Communication - User ID: 8 Security Level: 15 Password: root
IP address: 10.0.0.11 User Commands TCP Port: 1033
[ ] User Command Encryption [ ] History Rec. Encryption Key: abcdefghijk
[ ] Remonte UCC Connection UCC Database IP Address: 0.0.0.0 TCP Port: 3306

On-line Control Units [unit ID - uploaded translation]
Unit 1 from - Project version:1 - Translation no. 1 - Load attempt no.1
Unit 2 from - Project version:1 - Translation no. 1 - Load attempt no.8

Control Unit Elements (functions/variables/parameters) [ID - type - name]
56 - Binary Signal Generator - cfn:: bin.generator
57 - Physical Analog Input - cfn:: Physical AI32
58 - Physical AI Converter - cfn:: AI12 conv.N TC10k
59 - Physical AI Converter - cfn:: AI22 conv.N TC10k
60 - Physical AI Converter - cfn:: AI32 conv.N TC10k
61 - Physical AI Converter - cfn:: AI42 conv.N TC10k
62 - Physical AI Converter - cfn:: AI52 conv.N TC10k
63 - Physical AI Converter - cfn:: AI62 conv.N TC10k
64 - Physical AI Converter - cfn:: AI72 conv.N TC10k
65 - Physical AI Converter - cfn:: AI82 conv.N TC10k
66 - Constant Value - cfn:: Constant 2V
67 - Constant Value - cfn:: Constant 3V
68 - Constant Value - cfn:: Constant 4V

Translated Control Units Displays
Operations Over On-Line Unit Elements No. 62,
Process real-time experiment tools: Process Experiment Procedure

Figure 1: DCU Control Design Tool graphical interface main window.

Entire procedure of control design is not performed uniquely inside the graphical user interface of the toolbox. Three main interfaces are used for design. Firstly, the toolbox graphical user interface (see Figure 1) is used for project management, on-line monitoring, control system management. Secondly, Matlab Simulink or Scilab Xcos toolbox is used for programming control applications. Simulink and Xcos are dynamic simulators with comfortable interfaces for drawing and simulating diagrams of dynamic models and in our case control application diagrams. Thirdly, main command line window and all kind of available toolboxes of Matlab/Scilab may need be used for process experiments. For example, to prepare excitation signals, to analyze obtained process responses, etc. This third part of control design is optional and it is intended for experienced control systems engineers who require optimized and robust control system performance.

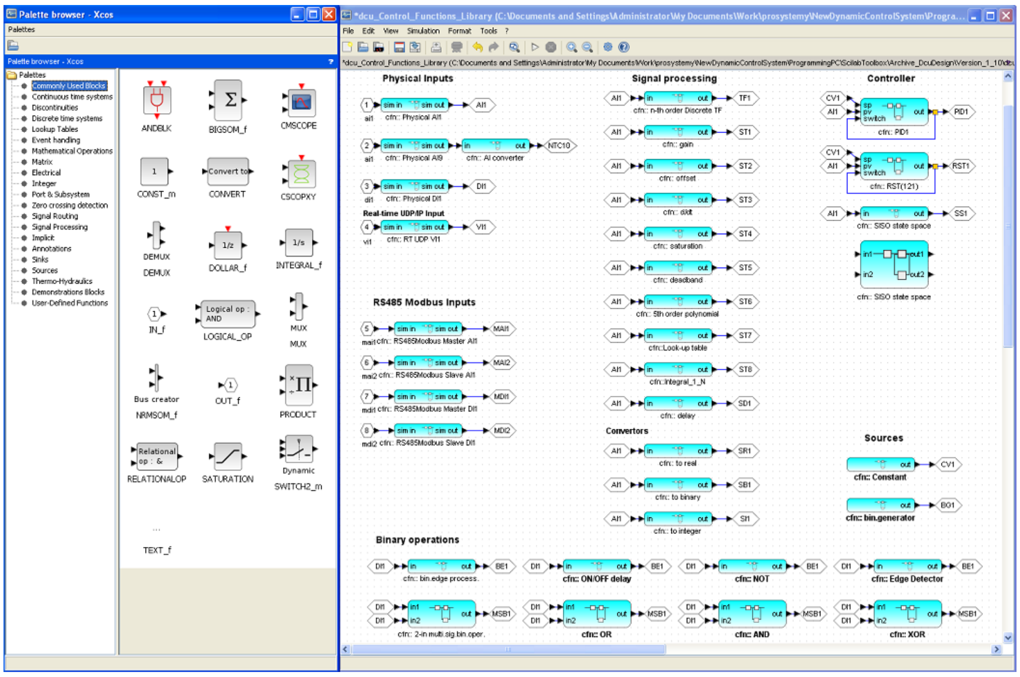


Figure 2: Xcos toolbox graphical interface.

Internal structure of control application diagram

Control application is a real-time program for DCU controller. DCU controls process according to loaded control application. Control application is programmed using functional diagram only. The functional diagram contains interconnected functional blocks and some pre-defined parameters. Control application for one DCU is defined inside super-block (group of blocks) which has at the beginning of its name “dcu::” string, see Figure. Main control application may contain multiple DCU super-blocks. To edit control application of a DCU, double-click the DCU super-block. The actual diagram of the corresponding super-block opens and user can start add, remove, edit control functions and its connections. All control functions are in library diagram that is in DCU design tool directory. Notice, that control function has always at the beginning of its name the string “cfn::”. It indicates to compiler that it is control function and not some simulation block.

Observing control application diagram, designer sees only DCU super-block, functional blocks inside DCU and its interconnections. Internal structure of control application functional diagram is however more complex, see Figure. Each functional block represent control function and control variable (only in case the function has a computed output value). Moreover, the block contains parameter definitions for the control function and for control variable. Finally, entire DCU super-block has parameters that defines general parameters of the control application. These are defined in context of the superblock (Scilab/Xcos) or in mask initialization (Matlab/Simulink).

Control application loaded into DCU consists of the following elements:

– Set of general parameters

– Control functions

– Control variables

General parameters defines general properties of control application such as used sampling times, DCU communication settings for UDP/TCP/IP and Modbus RTU, user accounts, etc. Control functions together with control variables, which are functions outputs or say results represent desired behavior of DCU outputs with respect to DCU inputs.

There is slight difference between the structure of these elements directly present inside DCU and how they are presented (for definition) to control system designer. The reason is to simplify designer's work. In principle, parameters that can be automatically evaluated from the control application functional diagram are not requested to be specifically defined by designer. Also storage of parameters inside DCU is much more compact than inside of the design tools. We will focus on control application from the designer point of view.

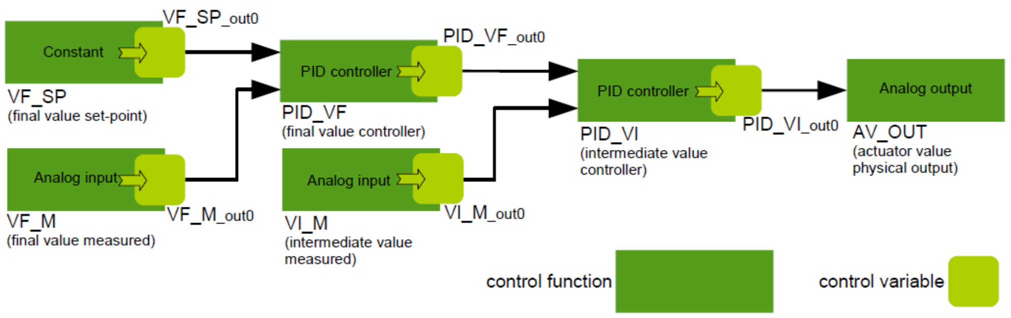
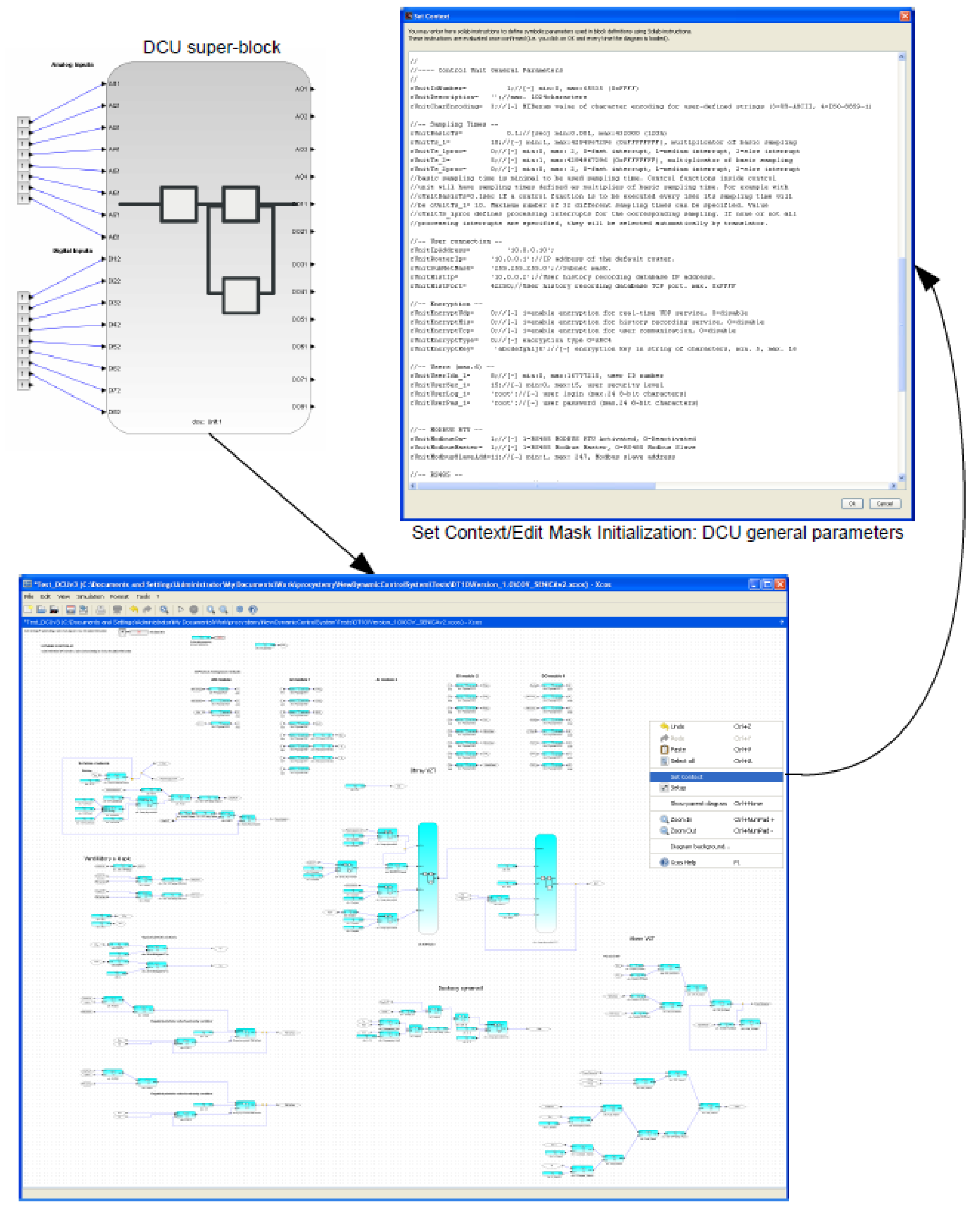


Figure : Internal structure of control application functional diagram.



You may enter here scilab instructions to define symbolic parameters used in block definitions using Scilab instructions.
These instructions are evaluated once confirmed (i.e. you click on OK and every time the diagram is loaded).

```
//
//---- Control Unit General Parameters
//
cUnitIdNumber=        1;//[-] min:0, max:65535 (0xFFFF)
cUnitDescription=   '';//max. 100characters
cUnitCharEncoding=  3;//[-] MIBenum value of character encoding for user-defined strings (3=US-ASCII, 4=ISO-8859-1)

//-- Sampling Times --
cUnitBasicTs=         0.1;//[sec] min:0.001, max:4294967 (1ms)
cUnitTs_1=          10;//[-] min:1, max:4294967295 (0xFFFFFFFF), multiplicator of basic sampling
cUnitTs_1proc=      0;//[-] min:0, max: 2, 0=fast interrupt, 1=medium interrupt, 2=slow interrupt
cUnitTs_2=          0;//[-] min:1, max:4294967295 (0xFFFFFFFF), multiplicator of basic sampling
cUnitTs_2proc=      0;//[-] min:0, max: 2, 0=fast interrupt, 1=medium interrupt, 2=slow interrupt
//basic sampling time is minimal to be used sampling time. Control functions inside control
//unit will have sampling times defined as multiplies of basic sampling time. For example with
//cUnitBasicTs=0.1sec if a control function is to be executed every 1sec its sampling time will
//be cUnitTs_1= 10. Maximum number of 32 different sampling times can be specified. Value
//cUnitTs_iproc defines processing interrupts for the corresponding sampling. If none or not all
//processing interrupts are specified, they will be selected automatically by translator.

//-- User connection --
cUnitIpAddress=        '10.0.0.10';
cUnitRouterIp=      '10.0.0.1';//IP address of the default router.
cUnitSubnetMask=    '255.255.255.0';//Subnet mask.
cUnitHistIp=        '10.0.0.2';//User history recording database IP address.
cUnitHistPort=      42100;//User history recording database TCP port. max. 0xFFFF

//-- Encryption --
cUnitEncryptRdp=    0;//[-] 1=enable encryption for real-time RDP service, 0=disable
cUnitEncryptHis=    0;//[-] 1=enable encryption for history recording service, 0=disable
cUnitEncryptUsr=    0;//[-] 1=enable encryption for user communication, 0=disable
cUnitEncryptType=   0;//[-] encryption type 0=RC4
cUnitEncryptKey=     'abcdefghij';//[-] encryption key is string of characters, min. 5, max. 16

//-- Users (max.4) --
cUnitUserIdn_1=     0;//[-] min:0, max:16777215, user ID number
cUnitUserSec_1=     15;//[-] min:0, max:15, user security level
cUnitUserLog_1=     'root';//[-] user login (max.24 8-bit characters)
cUnitUserPas_1=     'root';//[-] user password (max.24 8-bit characters)


//-- MODBUS RTU --
cUnitModbusOn=      1;//[-] 1=RS485 MODBUS RTU Activated, 0=Deactivated
cUnitModbusMaster=  1;//[-] 1=RS485 Modbus Master, 0=RS485 Modbus Slave
cUnitModbusSlaveAddr=1;//[-] min:1, max: 247, Modbus slave address

//-- RS485 --
```

Figure : DCU control application diagram.

Process experiment

A process experiment is an application of excitation signals onto process while recording process responses. Process experiment is defined by process variables to be set to constant override, process variables with dynamic override, external signals to be sent into process variables with dynamic override, process variables to be recorded while experiment is running to record process response. Example of process experiment is shown in Figure. User defines the experiment as described above and launches experiment. Control system then executes real-time experiment according to user specifications. An experienced Matlab/Scilab developer/researcher may include its own advanced control design technique programmed in Matlab/Scilab into DCU design tool and integrate process experiment into this technique. Example is provided in DCU design tool with Zigler-Nichols PID tuning method. This method needs to apply a step into open-loop process input and based on the process response the method computes coefficients of PID controller. The method is thanks to process experiment completely automatic. User only chooses PID controller to tune, selects amplitude and minimal length of step, and launch tuning.

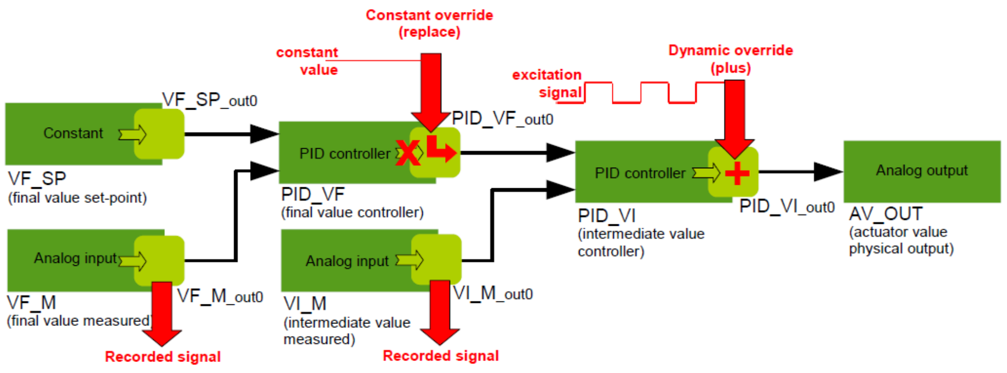


Figure: Example of process experiment.

Programming structure of the tool

The toolbox contains these main files:

dcu.sci (dcu.m) - contains function that initializes the tool and displays the main window of graphical user interface.

dcuCappDesTool_call.sci (dcuCappDesTool_call.m) - contains most of graphical user interface callbacks (functions called due to some actions made on the interface - press button etc.)

cfunsim.sci (cfunsim.m)- contains simulation code for some of the control functions. Control functions are function blocks of Xcos/Simulink for building control application diagram.

ucccall.sci (ucccall.m) - contains call of function for communicating with User Command Center (UCC). UCC is a Java application that communicates with DCU controllers and reads data from MySQL database.

dcu_library_x_y.zcos (dcu_library_x_y.mdl) - real-time control function library, where x and y defines version of the library (x.y). The file contains all types of control functions that can be used to build a control application to be loaded into DCU.

“ucccalls” directory - contains dcuUserCommandCall dynamic (shared) library and MySQL dynamic library for different operating system cores Windows 32-bit, Windows 64-bit, Mac OS X 64-bit, Linux 32-bit, Linux 64-bit. The library dcuUserCommandCall contains only one function. The function that is called from ucccall.sci (ucccall.m) file and that provides communication with UCC service.

The toolbox directory then contains some auxiliary function files, such as:

getcvar_actval.sci (getcvar_actval.m) - contains function that reads on-line from actually selected DCU actual value of a control variable.

getcfcn.sci (getcfcn.m) - contains function that returns control function structure

getcvar.sci (getcvar.m) - contains function that returns control variable structure

zieglernichols.sci (zieglernichols.m) - contains process experiment tool function that performs Ziegler-Nichols PID tuning over selected PID controller.

launchexperiment.sci (launchexperiment.m) - function that launches selected process experiment tool either in real-time or in simulation (according to simulation checkbox).

Data structure

All actual control design data are in one global structured variable called dcuData. After launching the design tool the variable is initialized into a default. If a project is loaded into DCU design tool, dcuData variable is overwritten with the loaded variable of the project. dcuData variable has the following structure:

```
dcuData=struct('mode',1,'nPrj',0,'prjs',[],'toolPath',x,'dbUser',    params(1),
'dbPassword',   params(2),   'prjPath','',   'prjName',   '',   'uccstatus',  0,
'nRtunit', 0, 'rtunits', [], 'nRtdisp', 0, 'rtdisps', [], 'rtdisptype', -1,
'nOnlinewin',  0,  'onlinewins',[],  'updatePeriod',  900,  'rtonlinemode',  1,
'dynOvrdBaseTcp', 49200, 'pc_ip', '10.0.0.2');
```

Project data

The field 'prjs' is a vector of versions of the project. Each version is defined with the following structure:

```
dcuPrj  =  struct('version',  params(1),  'file',[],  'path',[],  'nTransl',  0,
'transls', []);
```

The field 'transls' contains a vector of translations for the given project version. One translation is defined with the following structure:

newTransl = struct('path',[],'nDcu',0,'dcus',[],'date',[]);

The field 'dcus' contains a vector of translated controllers (DCUs) of the translation. One translated DCU is defined with the following structure:

```
newDcu = struct('id', 0, 'name', [], 'ip', '0.0.0.0', 'filename', '', 'userid',
0, 'usersec', 0, 'userpass', [], 'nFcn', 0, 'fcns', [], 'nVar', 0, 'vars', [],
'ucport',  1033,  'crypt',  0,  'ckey',  [],  'ubus',  0,  'nLoad',  0,  'loads',  [],
'remucc',  0,  'remip',  '0.0.0.0',  'remtcp',  3306,  'dbuser',  dcuData.dbUser,
'dbpass',  dcuData.dbPassword,  'hrecport',  0,  'pcport',  0,  'pcip',  '0.0.0.0',
'onpcsamp', -1, 'onpcid', 0, 'onpcsampinx', -1, 'nusedio', 0, 'maxoutsamp', 0);
```

The fields 'fcns' and 'vars' contain vectors of control functions and control variables. One control function is defined with the following structure:

```
newFcn = struct('id',0,'name',[],'type',[],'nout',0,'outs',[], 'nin', 0, 'ins',
[]);
```

One control variable is defined with the following structure:

```
newVar = struct('id',0,'name',[],'type',[]);
```

On-line data

In dcuData variable the field 'rtunits' contains vector of on-line DCU controllers, the controllers that has already been uploaded with a translated control application. One on-line DCU is defined by the structure with the following fields:

```
id: 1
ver: 1
transl: 1
dcu: 1
load: 87
nFcnexec: 136
fcnexecs: [136x1 constant]
nVarVal: 124
varVals: [124x1 constant]
histRecServ: -1
monitor: 0
onChipSimServ: -1
onChipSimMode: 0
actMode: -1
lastupdate: 0
basicSampling: 0
nAi: 16
physicalAi: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
nAo: 8
physicalAo: [0,0,0,0,0,0,0,0]
nDio: 32
physicalDio: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
physicalDioRx:
[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
onPcServ: -1
onPcMode: 0
onpcRwPeriod: -1
onpcLostFactor: 0
servedio: 0
syncdelay: 0
lastoutsample: 0
servedfcn: [0x0 constant]
```

Adding real-time process experiment tools

There are two variants of implementation for a new process experiment tools. Both are showed in DCU design tool. First variant (standard-interface implementation) uses standard process experiment specification window (see Figure) to define the experiment. By clicking on “Start Real-Time Process Experiment” button, the process experiment is launched. When the experiment is finished, processing tool is launched automatically afterward. The specification window is filling global structured variable SEspec that can be read by processing tool. The recorded process responses are stored not only in variables whose names are defined in the specification window but also in SEspec variable. The variable has the following structure:

```
SEspec =
struct('constOvrd',cOvrd,'dynOvrd',dOvrd,'recVar',rVar);
```

The field constOvrd is a vector of structured variables containing definitions of constant overrides. One constant override is defined by the following structure:

```
cOvrd(i)= struct('iver', project_version_index, 'itr',
translation_index, 'idcu', dcu_index, 'unitid', dcu_id, 'ivar',
varID, 'params',vecpars, 'sig','');
```

The vector of parameters 'vecpars' contains the following values: [override operation(1=replace, 2=plus, 3=multiply), override duration[sec], delay from beginning of the experiment[sec], override value]

The field dynOvrd is a vector of structured variables containing definitions of dynamic overrides. One dynamic override is defined by the following structure:

```
dOvrd(i)= struct('iver', project_version_idx, 'itr',
translation_idx, 'idcu', dcu_idx, 'unitid', dcu_id, 'ivar',
varID, 'params',vecpars, 'sig','dyn_ovrd_excitation_signal');
```

The vector of parameters 'vecpars' contains the following values: [override operation(1=replace, 2=plus, 3=multiply), delay from beginning of the experiment[sec]]. The field 'sig' contains the name of variable containing samples of dynamic override signal.

The field recVar is a vector of structured variables containing definitions of control variables to be recorded as process responses. One recorded variable is defined by the following structure:

```
rVar(i)= struct('iver', project_version_idx, 'itr',
translation_idx, 'idcu', dcu_idx, 'unitid', dcu_id, 'ivar',
varID, 'params',[], 'sig','recorded_signal', 'values',[],
'times',[]);
```

The fields 'values' and 'times' will appear only after the experiment. They will contain samples and sample times respectively of the recorded process response signal. The field 'sig' contains the name of variable that will as well contain the recorded signal. The variable will have the structure with two fields 'values' and 'times'.

Second implementation variant (specific-interface implementation) uses its own specific interface to get specification parameters needed for the process experiment tool. Process experiment has to be launched directly by the tool. A variable with the same structure as the global variable SEspec must be defined inside the tool, since this variable is an input parameter for the function launchexperiment, that launches process experiment.

The standard-interface implementation is shown on the implementation of the “Process Experiment Procedure” tool. The processing tool launched afterward is display of recorded process responses. The specific-interface implementation is employed for Ziegler-Nichols PID Tuning tool.

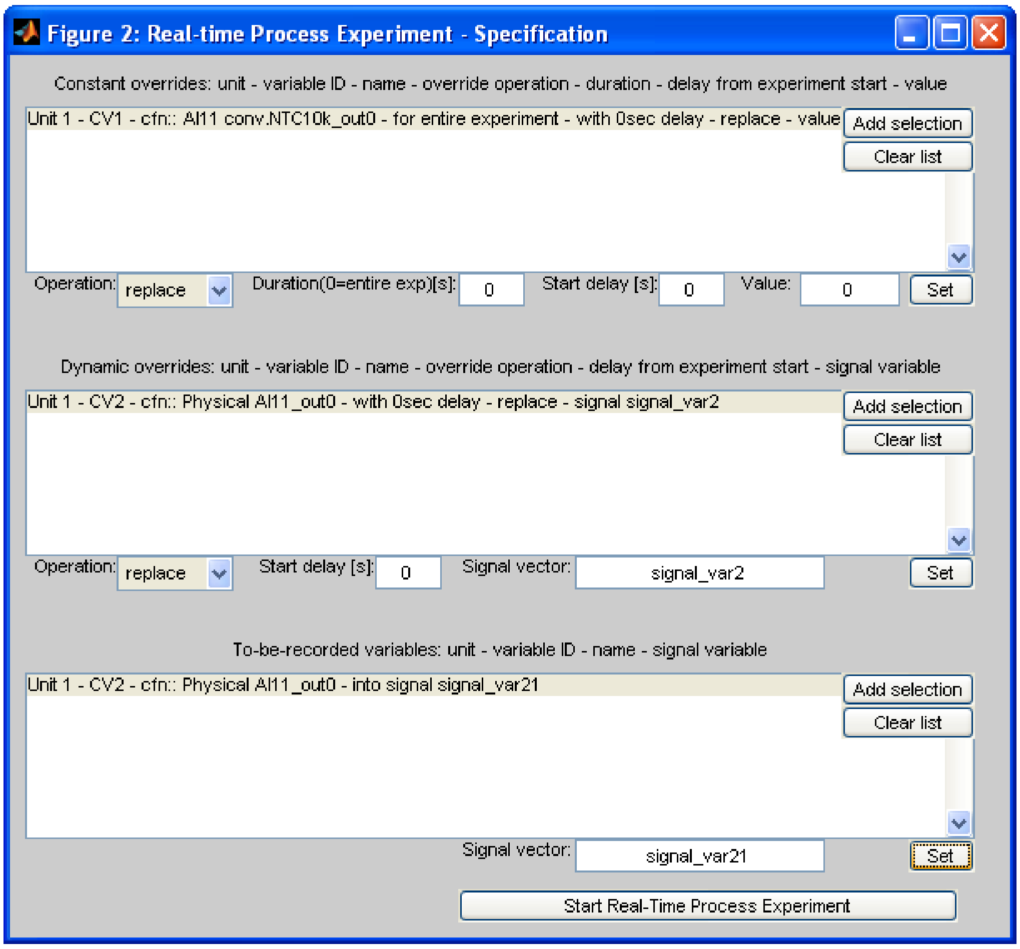


Figure 2: Real-time Process Experiment - Specification

Constant overrides: unit - variable ID - name - override operation - duration - delay from experiment start - value

Unit 1 - CV1 - cfn:: AI11 conv.NTC10k_out0 - for entire experiment - with 0sec delay - replace - value

Operation: replace; Duration(0=entire exp)[s]: 0; Start delay [s]: 0; Value: 0

Dynamic overrides: unit - variable ID - name - override operation - delay from experiment start - signal variable

Unit 1 - CV2 - cfn:: Physical AI11_out0 - with 0sec delay - replace - signal signal_var2

Operation: replace; Start delay [s]: 0; Signal vector: signal_var2

To-be-recorded variables: unit - variable ID - name - signal variable

Unit 1 - CV2 - cfn:: Physical AI11_out0 - into signal signal_var21

Signal vector: signal_var21

Figure: Standard process experiment specification window.

All process experiment tools are defined at the beginning of dcu.sci (dcu.m) file. A tool is defined by the following code:

```
peTool.title = 'Ziegler-Nichols PID tunning';
peTool.usepespecifwindow = 'no';
peTool.description = 'Ziegler-Nichols technique for PID
coefficints (K, Ti, Td, N) tuning.';
peTool.processingtool = 'zieglernichols(online_elements);';
peTools(x)=peTool;
```

where x is the index of the tool. The filed .title defines title of the tool that will appear in Process experiment tools selector.

The filed .usepespecifwindow defines implementation variant. 'yes' indicates standard-interface implementation, 'no' indicates specific-interface implementation.

The filed .description defines description of the tool. In case of standard-interface implementation, the description is displayed together with specification window. In case of specific-interface implementation the description is dispalyed before launching the tool. Developer can inform user about the tool behavior or what names must have the variables that hold recorded process responses and the variables that hold dynamic override signals so that the processing tool can find them.

The filed .processingtool defines calling command of the tool. In case of standard-interface implementation, the tool is called when the process experiment defined in the standard specification window is finished. In case of specific-interface implementation the tool is launched as soon as user confirms the tool launching. The calling command may contain variable named 'online_elements' (see the Ziegler-Nichols PID Tuning too) which contains actually selected on-line DCU elements.

Reference

[1] DCU Control System User Manual, Prosystemy, 2014 (available online at: www.prosystemy.sk)