Automating importing data into Matlab

# What you can do:

* Automatically import data from the bobcat database into matlab and C.
* Automatically import #define and enum symbols from C into matlab.

# Prerequisites

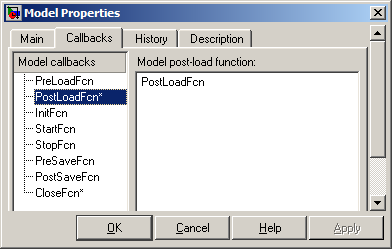
* Matlab 2007b or later (earlier versions may work)
* The Perl scripts available from SCM in Mathworks\_Utilities\Scripts
* The datbase\_reader.exe program (same location as above)
* You must have already set up the Bobcat custom storage classes as described in the mathworks best practices document. This procedure is under the heading *Matlab Configuration Settings*, steps 1-3.

# General setup

## The MATLAB workspace and PostLoadFcn

To get a script to run when your matlab project loads, you can add it to the PostLoadFcn callback under the Simulink model properties. This is described the *Mathworks Best Practices* document.

We will construct a master script for your project, called PostLoadFcn.m, and place a call to it in the PostLoadFcn dialog. PostLoadFcn.m will be used to populate the workspace with all the data imported from c and/or the bobcat database.



The script may also be called from the Matlab command line if the workspace becomes polluted (from loading another project, for example), or if source files have changed and the workspace needs to be refreshed. It should be located in your project’s working directory.

## locating the scripts

The perl scripts from (2) must be in your matlab path. If you’ve followed the matlab best practices document, then this should already be the case. BobcatUtils\_install.m will add Mathworks\_Utilities\Scripts to your path on startup. You may confirm this with the path command.

Alternatively, you could use a call to addpath in your startup.m script.

## Running a Perl script in MATLAB

The standard install of Matlab 2007b includes a Perl interpreter, version 5.0. It can be invoked from within matlab with the perl command. perl returns the output of the perl script run. The script name and arguments are passed as strings. The following example will display the help for the script read\_cd2.pl if typed into the command line:

perl('read\_cd2.pl', '-h')

The help text for each script mentioned here may be read in this way.

When printing output by omitting a semi-colon, Matlab will always add an “ans =”, which I find distracting. The example PostLoadFcn.m (see below) uses the disp function instead:

ret\_str\_tmp = perl('read\_cd2.pl', '-h');

disp(ret\_str\_tmp);

# Importing data from the bobcat database

You can use the read\_cd2.pl script for this purpose. It can read the contents of a database file (sadb file) and generate the corresponding c, h, and m files required to interface with your code.

## read\_cd2.pl

Run read\_cd2.pl –h for a description of all possible arguments. The following is a sample invocation.

perl ('read\_CD2.pl', '-C',...

'-a', 'cdata,0x4200',...

'-o', '..\sources\cdata.h',...

'-p', '..\sources\cdata.c',...

'-q', 'cdata.m',...

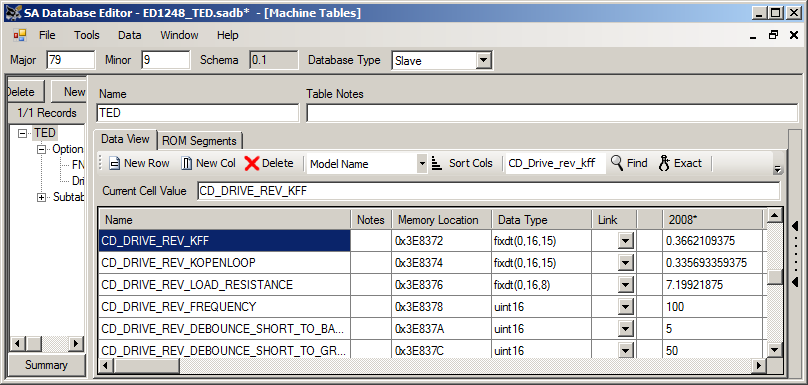
'c:\projects\service analyzer\experimental\_release\ED1248\_TED.sadb,

'TED');

This example is from the NextGen Telehandler Drive, whose database name is TED. Note that all relative paths are relative to your matlab working directory. Options o,p,q specify the locations of the output files. Option a forces the location of the main table (named cdata) in cdata.c to 0x4200 (a near address, which differs from the absolute one in the database). Option C skips building if the timestamp on the sadb file has not changed.

## Specifying data types in the database

The data types from the editor should be respected by the script. For example:



CD\_DRIVE\_REV\_KFF will be generated as the following in cdata.m:

CD\_CD\_DRIVE\_REV\_KFF = Bobcat.Parameter;

CreateAliasCD( CD\_CD\_DRIVE\_REV\_KFF,fi(0,0,16,15,'hex','2ee0'),...

'ConstantData' ,'CONSTANT\_DATA' ,'CD\_DRIVE\_REV\_KFF');

In cdata.h, CD\_DRIVE\_REV\_KFF appears as:

uint16\_t CD\_DRIVE\_REV\_KFF; //fixdt\_0\_16\_15

If instead CD\_DRIVE\_REV\_KFF had been specified as fixdt(1,16,8), then it would appear in cdata.h as:

int16\_t CD\_DRIVE\_REV\_KFF; //fixdt\_1\_16\_8

# Importing #define and enum symbols from C

There are two perl scripts available to automatically generate m-files for symbols defined in your project header files. They are read\_enums.pl and read\_defines.pl. The former will read enumerations, the latter “#define”s.

## An Example

This example generates matlab symbols from the defines.h, fault\_table.h, and ap\_dv.h files:

perl ('read\_defines.pl',...

'-o', 'defines.m', '../sources/defines.h',...

'../sources/fault\_table.h', '../sources/ap\_dv.h') ;

perl ('read\_enums.pl',...

'-o', 'enums.m', '../sources/defines.h',...

'../sources/fault\_table.h', '../sources/ap\_dv.h');

Option –o specifies the output file.

## A warning

These scripts are not as capable as your compiler. The will not follow #include directives. They may choke on particularly complicated declarations. If you need to hide parts of a header file that these scripts will not understand, you can include ‘MATLAB\_STOP\_READING’ in a comment. After this string is encountered, the rest of the file will be skipped.

## Specifying data types

You may specify the fixed-point type that all enumerated or #defined data is imported as with the ‘–t’ flag. It is also possible to specify a data type for an individual #define statement with //fixdt\_0\_16\_8 or //uint16 comments. For example:

#define ACTIVE 1 //uint8

ANSI-C specifies that the data type of an enumeration is int, so setting your enumerated data types to the word size of your processor is a good rule of thumb.

perl ('read\_defines.pl',...

'-t', 'uint16','-o', 'defines.m',...

'../sources/defines.h','../sources/fault\_table.h',...

'../sources/ap\_dv.h');

perl ('read\_enums.pl',...

'-t','int16', '-o','enums.m',...

'../sources/defines.h','../sources/fault\_table.h',...

'../sources/ap\_dv.h');

# Importing global variables

Sorry, but this is not implemented (and is unlikely to be implemented). For procedures on how to import EEPROM variables (and other global variables) via matlab m-files, see the Matlab best practices document.

# Appendix 1: Example PostLoadFcn.m

clear; %wipe out the current workspace

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%import data from the bobcat database

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%note - this script also updates cdata.h and cdata.c

try

ret\_str\_tmp = perl ('read\_CD2.pl', '-C',...

'-a', 'cdata,0x4200',...

'-o', '..\sources\cdata.h',...

'-p', '..\sources\cdata.c',...

'-q', 'cdata.m',...

'c:\projects\service analyzer\experimental\_release\ ED1248\_TED.sadb ',...

'TED');

disp(ret\_str\_tmp);

catch ME

disp(ME.message);

disp('PostLoadFcn.m failed to run read\_CD2.pl. cdata.m has not been updated.');

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%import enum & #define symbols

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

try

ret\_str\_tmp = perl ('read\_defines.pl',...

'-t','uint16', '-o','defines.m',...

'../sources/defines.h','../sources/fault\_table.h', '../sources/ap\_dv.h');

disp(ret\_str\_tmp);

catch ME

disp(ME.message);

disp('PostLoadFcn.m failed to run read\_defines.pl. defines.m has not been updated.');

end

try

ret\_str\_tmp = perl ('read\_enums.pl',...

'-t','int16', '-o','enums.m',...

'../sources/defines.h','../sources/fault\_table.h', '../sources/ap\_dv.h');

disp(ret\_str\_tmp);

catch ME

disp(ME.message);

disp('PostLoadFcn.m failed to run read\_enums.pl. enums.m has not been updated.');

end

clear ret\_str\_tmp;

clear ME;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%run the auto-generated m-files

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

cdata;

defines;

enums;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%m-files built by hand

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

misc\_defs;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%model sample rate

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

srate = 0.01;

# Appendix 2: script/executable documentation

Note: run each script/executable with ‘-h’ for the most recent documentation.

## read\_cd2.pl

Generates the C, H, and M files necessary for linking application code with

an SADB database.

usage: read\_CD2.pl [<flags>] <SADB file> <main table name>

-a <name>,<address> set the struct/array <name>'s address here, ignore

address in database, remove far qualifier (metrowerks only)

-e <offset> set the address of all structs/arrays relative to this offset,

remove far qualifier (metrowerks only)

-b <bytes> Break structures in two if the gap between consecutive symbols

exceeds <bytes>.

-d <default column name> use this default column in the machine table.

If not specified, the first default column is used.

-f <name>,<class> wrap the struct/array <name> in '#pragma class nc=<class>'

instead of using the \_at() qualifier, declare it as \_near (tasking only)

-m <machine column name> use this machine table column. If not specified,

only the default column will be used.

-n add a number suffix to variables that appear in a table more than once

-o <output h file name> use this output file name (default cdata.h)

-p <output c file name> use this output file name (default cdata.c)

-q <output m file name> use this output file name (default cdata.m)

-r <name> use this main table name in generated code (default cdata)

Note: when using -b, <name> may be a comma-separated list.

-t add a time stamp to the output file headers

-A prepend 'CD\_' to all main table variables in matlab with an alias

-C skip build if database time stamp has not changed (generates \*.make file)

-T generate C code for the tasking compiler (default is for metrowerks)

-W <warning num list> ignore these warning numbers (ex: -W 4,1,2)

note: lists of arguments must be separated by commas, with no spaces.

\* Sub-tables will be imported as arrays if their members are of uniform size,

and as structures otherwise.

\* The machine table will always be imported as a structure.

\* Byte arrays will be inserted into any gaps found in structures.

\* If a link table name is preceded by the machine's name, the machine's name

will be omitted from the symbol name in the output files.

Warnings:

1 could not open <file>.make

4 <name> in <struct/array> overlaps <name> in <struct/array> at address

<address>

10 table <name> is being removed because it has no rows

21 the elements of array <name> are misaligned (starts with element <num>)

22 element <num> of array <name> is being removed because it has an invalid

address <address>

23 element <num> of array <name> has no value

24 element <num> of table <name> has a non-numeric value

31 symbol <name> in struct <name> occurs more than once

32 symbol <name> is being removed because it overlaps previous symbol

<name> in struct <name>

34 symbol <name> in struct <name> has no value

35 symbol <name> in struct <name> has a non-numeric value

36 symbol <name> in struct <name> (<num> bytes) occurs at an odd

address

## read\_enums.pl

Generates an m-file from a c header file that contains all the 'enum'

symbols therein. Processing of a file's contents stops if

'MATLAB\_STOP\_READING' is encountered.

WARNING: this script is not a compiler! It does not follow #include

directives. Complicated definitions may confuse it.

usage: read\_enums.pl [-o <output file name>] <input header files...>

-o <output file name> use this output file name (default enums.m)

-t <type> cast all matlab variables to <type>. <type> may be of the form

'uint16', 'fixdt\_0\_16\_8', or variations thereof.

## read\_defines.pl

Generates an m-file from a c header file that contains all the '#define'

symbols therein. Processing of a file's contents stops if

'MATLAB\_STOP\_READING' is encountered.

WARNING: this script is not a compiler! It does not follow #include

directives. Complicated definitions may confuse it.

usage: read\_defines.pl [<flags>] <input header files...>

-o <output file name> use this output file name (default defines.m)

-t <type> cast all matlab variables to <type>. <type> may be of the form

'uint16', 'fixdt\_0\_16\_8', or variations thereof.

'#define's followed by //fixdt\_0\_16\_8 or //int8, etc will be cast to their

respective data types. This overrides the -t option.