eCMD Perl Module Reference Manual

Generated by Doxygen 1.2.15

Fri Jun 11 08:14:53 2004

Contents

Chapter 1

eCMD Perl Module Main Page

1.1 eCMD Perl API

In addition to a C/C++ API, eCMD provides a Perl API. To do this, Perl's DynaLoader module handles the loading of a special eCMD shared object and the Perl XS interface handles the function calls between eCMD C code and Perl.

The API interface is implemented as a "**ecmdClientPerlapi** (p. 12)" object. The **ecmdClientPerlapi** (p. 12) class has methods to initialize eCMD, make standard function calls. With the Perl API, scan data is passed with binary character strings between the client Perl script and the shared object's **ecmdClientPerlapi** (p. 12) class.

1.2 Packages

To load the eCMD shared object, Perl scripts should *use* (or *require*) the Perl package appropriate for the desired environment (Aix or Linux). Keep in mind the ECMD_DLL_FILE env variable must be set to the appropriate eCMD Plugin that you would like to run.

It is possible to use the *require* command in Perl to dynamically load the correct module for the emvironment you are running in, so your code will work across multiple platforms. For example:

```
#!/usr/bin/perl

@filestat = stat("/usr/include/linux/zorro.h");
if (@filestat[1] eq "") {
  require ecmd_aix_perl;
} else {
  require ecmd_x86_perl;
}
```

1.3 Perl Version

Scripts need Perl version 5.6.1. eCMD provides the proper version for both the AIX and Linux in all our supported sites and remote installs. All scripts using the perl module should start with

the following:

```
#!/bin/ksh
#! -*- perl -*-

eval '
if [ "X$CRONUS_HOME" = "X" ]; then echo "CRONUS_HOME env var is not set, rerun your Cronus setup script"; exit
PERL5LIB=$CRONUS_HOME/contrib/`uname`/perl/lib:$CRONUS_HOME/contrib/`uname`/perl/lib/5.6.1/osdep:$CRONUS_HOME/
export PERL5LIB;
exec $CRONUS_HOME/contrib/`uname`/perl/bin/perl -x -S $0 ${1+"$@"}
'
if 0;
```

This picks the proper perl based upon if you're running AIX or Linux and runs our copy in CRONUS_HOME.

1.4 Data Passing

Data is passed between the client Perl script and the eCMD shared object in the form of binary character strings. To make data manipulation in Perl much easier we have provided the **ecmdDataBuffer** (p. 34) Perl module. This allows you to have the same bit manipulation functions that are available on the C++ Api.

Here is an example usage of the **ecmdDataBuffer** (p. 34) module:

```
my $ep = new ecmdClientPerlapi();
if ($ep->initDll("")) { die "Fatal errors initializing DLL"; }

# Create a pointer to an DataBuffer class
my $data = new ecmdDataBuffer();
# Set the first word of data in this class
$data->setWord(0,0xFEEDBEEF);

# Read data from the chip
my $rc = 0;
# The $$ tells perl that we are passing in the class object by reference such that data can be returned out
# The general rule is if the function is defined as char** in the Api documentation pass in your variable as
# If the function is defined as const char* then pass it in as $$ as seen below.
$rc = $ep->getRing("pu -p1", "idreg", $$data);

# What is in the first word
print("Data : $data->getWord(0));

Change the value
$data->setWord(0,0xAAAA5555);
print("Data : $data->getWord(0));

# Write my new value to the chip
$rc = $ep->putRing("pu -p1", "idreg", $$data);
```

1.5 Chip/Object targeting

Most functions take a const char* i_target argument which is where you specify the target that you would like to operate on. There are two ways to do this, you can pass in the target as a string that is formatted as it would be on the command line (ie "pu -k2 -s3 -p2") or you can use the **ChipTarget** (p. 9) perl module provided. This allows for easier manipulation of the target you are operating on.

```
my $target = newChipTarget("pu -p2");
# The following two calls are equivalent
$rc = $ep->putRing("pu -p2", "idreg", $$data);
$rc = $ep->putRing($$target, "idreg", $$data);

# Now I can go to the next position
$target->pos(3);
$rc = $ep->putRing($$target, "idreg", $$data);
```

1.6 Perl API Usage

The following should be observed when using the Perl API.

1. The client should instantiate a new **ecmdClientPerlapi** (p. 12) object.
2. The initDll() function should ALWAYS be the first function called.
3. Data is sent from Perl to Cronus via strings for functions like putalias, putscom, etc.
4. Perl should be picked up as shown in the example below to grab the eCMD supported version

1.7 Example Perl Script

```
#!/bin/ksh
#! -*- perl -*-

eval '
if [ "X$CRONUS_HOME" = "X" ]; then echo "CRONUS_HOME env var is not set, rerun your Cronus setup script"; exit
PERL5LIB=$CRONUS_HOME/contrib/`uname`/perl/lib:$CRONUS_HOME/contrib/`uname`/perl/lib/5.6.1/osdep:$CRONUS_HOME/
export PERL5LIB;
exec $CRONUS_HOME/contrib/`uname`/perl/bin/perl -x -S $0 ${1+"$@"}
'
if 0;

# Grab the additional eCMD Modules;
require  ecmdDataBuffer  (p. 34);
require  ChipTarget  (p. 9);

my $filestat = stat("/usr/include/linux/zorro.h");
if ($filestat eq "") {

  require ecmd_aix_perl;
} else {
  require ecmd_x86_perl;
```

```
}

$| = 1;  # set autoflush

my $ep = new ecmdClientPerlapi();
if ($ep->initDll("")) { die "Fatal errors initializing DLL"; }


# Setup my target
my $target = new ChipTarget (p.9)("pu");
$target->pos(0);
$target->core(0);
$target->node(0);

# Setup my data
my $data = new ecmdDataBuffer();

my $rc = 0;

# put/getscom
if (0) {
   print "---- starting putscom -----";
   $data->setWord(0, 0xabebfeef);
   $rc = $ep->putScom($$target, "800000", $$data);

   print "---- starting getscom -----";
   $rc = $ep->getScom($$target, "800000", /$$data);
   print "getscom 800000 = " . sprintf("%0.8X %0.8X\\n", $data->getWord(0), $data->getWord(1));
}

# put/getring
if (0) {

   print "---- starting putring -----";
   $data->flushTo0();
   $data->setBit(0, 10); # Set bits 0-10

   $rc = $ep->putRing($$target, "testring", $$data);
   if ($rc) {
      print "Problems calling eCMD putring

";
   }
   print "---- starting getRing -----";
   $rc = $ep->getRing($$target, "testring", $$data);
   print "getRing string = $$data->getWord(0)\n";

}


$ep->cleanup();
```

Chapter 2

eCMD Perl Module Compound Index

2.1 eCMD Perl Module Compound List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

eCMD Perl Module File Index

3.1 eCMD Perl Module File List

Here is a list of all files with brief descriptions:

Chapter 4

eCMD Perl Module Class Documentation

4.1 ChipTarget Class Reference

Perl Class to hold eCMD Targetting information.

`#include <ChipTarget.H>`

Public Methods

- void **cage** (int i_cage)

  Set the Cage.

- void **node** (int i_node)

  Set the Node.

- void **slot** (int i_slot)

  Set the Slot.

- void **chip** (const char *i_chipname)

  Set the chip name.

- void **pos** (int i_pos)

  Set the chip position.

- void **core** (int i_core)

  Set the chip core.

- void **thread** (int i_thread)

  Set the chip thread.

4.1.1 Detailed Description

Perl Class to hold eCMD Targetting information.

Usage :

```
require ChipTarget;
my $target = new ChipTarget("pu -p1");
$target->node(2);
```

4.1.2 Member Function Documentation

4.1.2.1 void ChipTarget::cage (int *i_cage*)

Set the Cage.

Parameters:
i_cage New Cage Value

Usage : $target->cage(3);

4.1.2.2 void ChipTarget::node (int *i_node*)

Set the Node.

Parameters:
i_node New Node Value

Usage : $target->node(2);

4.1.2.3 void ChipTarget::slot (int *i_slot*)

Set the Slot.

Parameters:
i_slot New Slot Value

Usage : $target->slot(0);

4.1.2.4 void ChipTarget::chip (const char * *i_chipname*)

Set the chip name.

Parameters:
i_chipname New Chipname

Usage : $target->chip("pu");

4.1.2.5 void ChipTarget::pos (int *i_pos*)

Set the chip position.

Parameters:
i_pos New Chip Position

Usage : $target->pos(1);

4.1.2.6 void ChipTarget::core (int *i_core*)

Set the chip core.

Parameters:
i_core New Core Value

Usage : $target->core(1);

4.1.2.7 void ChipTarget::thread (int *i_thread*)

Set the chip thread.

Parameters:
i_thread New Thread Value

Usage : $target->thread(0);

The documentation for this class was generated from the following file:

- **ChipTarget.H**

4.2 ecmdClientPerlapi Class Reference

eCMD Perl API.

`#include <ecmdClientPerlapi.H>`

Public Methods

Load/Unload Functions

- **ecmdClientPerlapi** ()
  eCMD Perl Module Constructor.
- **~ecmdClientPerlapi** ()
  eCMD Perl Module Destructor.
- int **initDll** (const char *i_dllName, const char *i_options=NULL)
  Initialize the eCMD DLL.
- void **cleanup** ()
  Clean up the Perl Module.
- int **ecmdCommandArgs** (char **i_argv[])
  Pass any unknown command line paramaters to the DLL for processing (ex. -p#, Cronus -debug).

Chip Scan Functions

- int **getRing** (const char *i_target, const char *i_ringName, char **o_data)
  Scans the selected number of bits from the selected position in the selected ring into the data buffer.
- int **putRing** (const char *i_target, const char *i_ringName, const char *i_data)
  Scans the selected number of bits from the data buffer into the selected position in the selected ring.

Chip Scom Functions

- int **getScom** (const char *i_target, int i_address, char **o_data)
  Scoms bits from the selected address into the data buffer.
- int **putScom** (const char *i_target, int i_address, const char *i_data)
  Scoms bits from the data buffer into the selected address.

Jtag Functions

- int **sendCmd** (const char *i_target, int i_instruction, int i_modifier, char **o_status)
  Send a JTAG instruction and modifier to the specified chip.

FSI Functions

- int **getCfamRegister** (const char *i_target, int i_address, char **o_data)
  Read data from the selected CFAM register address into the data buffer.

- int **putCfamRegister** (const char *i_target, int i_address, const char *i_data)
  Write data into the selected CFAM register address.

Spy Functions

- int **getSpy** (const char *i_target, const char *i_spyName, char **o_data)
  Reads the selected spy into the data buffer.

- int **getSpyEnum** (const char *i_target, const char *i_spyName, char **o_enumValue)
  Reads the selected spy and returns it's assocaiated enum.

- int **getSpyEccGrouping** (const char *i_target, const char *i_spyEccGroupName, char **o_groupData, char **o_eccData, char **o_eccErrorMask)
  Read an ECC grouping and return the in and out bits as well as a error mask if any out bits are invalid.

- int **putSpy** (const char *i_target, const char *i_spyName, const char *i_data)
  Writes the data buffer into the selected position in the selected spy.

- int **putSpyEnum** (const char *i_target, const char *i_spyName, const char *i_enumValue)
  Writes the enum into the selected position in the selected spy.

Ring Cache Functions

- void **ecmdEnableRingCache** ()
  Enables internal caching of read/writes of scan rings to the chip for functions like getring/getspy/getspr.

- int **ecmdDisableRingCache** ()
  Disable internal caching of reads/writes of scan rings.

- int **ecmdFlushRingCache** ()
  Flush all modified data from the internal cache to the hardware, then remove all rings from cache.

Array Functions

- int **getArray** (const char *i_target, const char *i_arrayName, const char *i_address, char **o_data)
  Reads bits from the selected array into the data buffer.

- int **putArray** (const char *i_target, const char *i_arrayName, const char *i_address, const char *i_data)
  Writes bits from the data buffer into the selected array.

Simulation Functions

- int **simaet** (const char *i_function)
  Enable/Disable Simulation AET Logging.

- int **simcheckpoint** (const char *i_checkpoint)
  Store a checkpoint to specified file.

- int **simclock** (int i_cycles)
  Clock the model.

- int **simecho** (const char *i_message)
  Echo message to stdout and sim log.

- int **simexit** ()
  Close down the simulation model.

- int **simEXPECTFAC** (const char *i_facname, int i_bitlength, const char *i_expect, int i_row=0, int i_offset=0)
  Perform expect on facility using name.

- int **simexpecttcfac** (const char *i_tcfacname, int i_bitlength, const char *i_expect, int i_row=0)
  Perform expect on TCFAC facility.

- int **simgetcurrentcycle** (char **o_cyclecount)
  Fetch current model cycle count.

- int **simGETFAC** (const char *i_facname, int i_bitlength, char **o_data, int i_row=0, int i_offset=0)
  Retrieve a Facility using a name.

- int **simGETFACX** (const char *i_facname, int i_bitlength, char **o_data, int i_row=0, int i_offset=0)
  Retrieve a Facility using a name - preserving Xstate.

- int **simgettcfac** (const char *i_tcfacname, char **o_data, int i_row=0, int i_startbit=0, int i_bitlength=0)
  Retrieve a TCFAC facility.

- int **siminit** (const char *i_checkpoint)
  Initialize the simulation.

- int **simPUTFAC** (const char *i_facname, int i_bitlength, const char *i_data, int i_row=0, int i_offset=0)
  Write a Facility using a name.

- int **simPUTFACX** (const char *i_facname, int i_bitlength, const char *i_data, int i_row=0, int i_offset=0)
  Write a Facility using a name - preserving Xstate.

- int **simputtcfac** (const char *i_tcfacname, int i_bitlength, const char *i_data, int i_row=0, int i_numrows=0)
  Write a TCFAC facility.

- int **simrestart** (const char *i_checkpoint)
  Load a checkpoint into model.

- int **simSTKFAC** (const char *i_facname, int i_bitlength, const char *i_data, int i_row=0, int i_offset=0)
  Stick a Facility using a name.

- int **simstktcfac** (const char *i_tcfacname, int i_bitlength, const char *i_data, int i_row=0, int i_numrows=0)
  Stick a TCFAC facility.

- int **simSUBCMD** (const char *i_command)
  Run RTX SUBCMD.

- int **simtckinterval** (int i_tckinterval)
  Set TCK Interval setting in the model for JTAG Master.

- int **simUNSTICK** (const char *i_facname, int i_bitlength, int i_row=0, int i_offset=0)
  Unstick a Facility using a name.

- int **simunsticktcfac** (const char *i_tcfacname, int i_bitlength, const char *i_data, int i_row=0, int i_numrows=0)
  Unstick a TCFAC facility.

Error Handling Functions

- char * **ecmdGetErrorMsg** (int i_errorCode)
  Retrieve additional error information for errorcode.

Output Functions

- void **ecmdOutputError** (const char *i_message)
  Output a message related to an error.

- void **ecmdOutputWarning** (const char *i_message)
  Output a message related to an warning.

- void **ecmdOutput** (const char *i_message)
  Output a message to the screen or logs.

4.2.1 Detailed Description

eCMD Perl API.

Usage :

```
@filestat = stat("/usr/include/linux/zorro.h");
if (@filestat[1] eq "") {
  require ecmd_aix_perl;
} else {
  require ecmd_x86_perl;
}
my $ep = new  ecmdClientPerlapi()  (p. 16);
```

4.2.2 Constructor & Destructor Documentation

4.2.2.1 ecmdClientPerlapi::ecmdClientPerlapi ()

eCMD Perl Module Constructor.

4.2.2.2 ecmdClientPerlapi::~ecmdClientPerlapi ()

eCMD Perl Module Destructor.

4.2.3 Member Function Documentation

4.2.3.1 int ecmdClientPerlapi::initDll (const char * *i_dllName*, const char * *i_options* = NULL)

Initialize the eCMD DLL.

Return values:
0 if successful load 1 if unsuccessful

Parameters:
i_dllName Full path and filename of the eCMD Dll to load

i_options Passed from Command line Arguments

Precondition:
ecmdClientPerlapi constructor must have been called

Postcondition:
Global options (ex. -p#, -c#) will be removed from arg list

- initializes the eCMD Dll.
- Global options such as -p#, -c# will be parsed out.
- Position flags can be queried later with functions like waspositionselected()

4.2.3.2 void ecmdClientPerlapi::cleanup ()

Clean up the Perl Module.

4.2.3.3 int ecmdClientPerlapi::ecmdCommandArgs (char ** *i_argv*[])

Pass any unknown command line paramaters to the DLL for processing (ex. -p#, Cronus -debug).

Return values:
ECMD_SUCCESS if successful

nonzero if unsuccessful

Parameters:
i_argv Passed from Command line Arguments

Precondition:
loadDll must have been called

Postcondition:
Global options (ex. -debug, -p#, -c#) will be removed from arg list

See also:
loadDll

- argc/argv get passed to the eCMD DLL.
- Global options such as -debug flags and -p#, -c# will be parsed out.
- Position flags can be queried later with functions like ????? NOTE : This function does not affect ring caching

4.2.3.4 int ecmdClientPerlapi::getRing (const char * *i_target*, const char * *i_ringName*, char ** *o_data*)

Scans the selected number of bits from the selected position in the selected ring into the data buffer.

Return values:
ECMD_INVALID_RING if ringname is not valid for target

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_SUCCESS if successful

nonzero if unsuccessful

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

Parameters:
i_target Struct that contains chip and cage/node/slot/position/core information of ring to read

i_ringName Name of ring to read from

o_data DataBuffer object that holds data read from ring

See also:
putRing (p. 17)

4.2.3.5 int ecmdClientPerlapi::putRing (const char * *i_target*, const char * *i_ringName*, const char * *i_data*)

Scans the selected number of bits from the data buffer into the selected position in the selected ring.

Return values:
ECMD_SUCCESS if successful

nonzero if unsuccessful

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_RING if ringname is not valid for target

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core information of ring to write

i_ringName Name of ring to write to

i_data DataBuffer object that holds data to write into ring

See also:

getRing (p. 17)

4.2.3.6 int ecmdClientPerlapi::getScom (const char * *i_target*, int *i_address*, char ** *o_data*)

Scoms bits from the selected address into the data buffer.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core information of scom address to read

i_address Scom address to read from

o_data DataBuffer object that holds data read from address

See also:

putScom (p. 18)

4.2.3.7 int ecmdClientPerlapi::putScom (const char * *i_target*, int *i_address*, const char * *i_data*)

Scoms bits from the data buffer into the selected address.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core information of scom address to write

i_address Scom address to write to

i_data DataBuffer object that holds data to write into address

See also:

getScom (p. 18)

4.2.3.8 int ecmdClientPerlapi::sendCmd (const char * *i_target*, int *i_instruction*, int *i_modifier*, char ** *o_status*)

Send a JTAG instruction and modifier to the specified chip.

Parameters:

i_target Struct that contains chip and cage/node/slot/position information of scom address to write

i_instruction Right aligned instruction to send to chip

i_modifier Right aligned instruction modifier to send

o_status Instruction status register value retrieved

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

ECMD_NON_JTAG_CHIP Chip Target is a non-jtag attached chip

nonzero if unsuccessful

NOTE : Proper parity will be generated on the command and modifier

4.2.3.9 int ecmdClientPerlapi::getCfamRegister (const char * *i_target*, int *i_address*, char ** *o_data*)

Read data from the selected CFAM register address into the data buffer.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

nonzero if unsuccessful

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

Parameters:

i_target Struct that contains chip and cage/node/slot/position information

i_address CFAM address to read from

o_data DataBuffer object that holds data read from address

4.2.3.10 int ecmdClientPerlapi::putCfamRegister (const char * *i_target*, int *i_address*, const char * *i_data*)

Write data into the selected CFAM register address.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position information

i_address CFAM address to write to

i_data DataBuffer object that holds data to write into address

4.2.3.11 int ecmdClientPerlapi::getSpy (const char * *i_target*, const char * *i_spyName*, char ** *o_data*)

Reads the selected spy into the data buffer.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SPY_FAILED_ECC_CHECK if invalid ECC detected on Spy read

ECMD_INVALID_SPY Spy name is invalid or Spy is an ECC Grouping

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_SPY_IS_EDIAL Spy is an edial have to use getSpyEnum

ECMD_SPY_GROUP_MISMATCH A mismatch was found reading a group spy not all groups set the same

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core information of spy to read

i_spyName Name of spy to read from

o_data DataBuffer object that holds data read from spy

NOTE : This function is ring cache enabled

4.2.3.12 int ecmdClientPerlapi::getSpyEnum (const char * *i_target*, const char * *i_spyName*, char ** *o_enumValue*)

Reads the selected spy and returns it's assocaiated enum.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SPY_FAILED_ECC_CHECK if invalid ECC detected on Spy read - valid Spy Data still returned

ECMD_INVALID_SPY Spy name is invalid or Spy is an ECC Grouping

ECMD_INVALID_SPY_ENUM if value in hardware doesn't map to a valid enum

ECMD_SPY_NOT_ENUMERATED Spy is not enumerated must use getSpy

ECMD_SPY_GROUP_MISMATCH A mismatch was found reading a group spy not all groups set the same

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_SUCCESS if successful read

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core information of spy to read

i_spyName Name of spy to read from

o_enumValue Enum value read from the spy

NOTE : This function is ring cache enabled

4.2.3.13 int ecmdClientPerlapi::getSpyEccGrouping (const char * *i_target*, const char * *i_spyEccGroupName*, char ** *o_groupData*, char ** *o_eccData*, char ** *o_eccErrorMask*)

Read an ECC grouping and return the in and out bits as well as a error mask if any out bits are invalid.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

ECMD_INVALID_SPY Spy name is invalid or Spy is not an ECC Grouping

ECMD_SPY_FAILED_ECC_CHECK if invalid ECC detected on Spy read - valid Spy Data still returned

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core information of spy to read

i_spyEccGroupName Name of spy to read from

o_groupData Return the data for the input to the eccGroup

o_eccData Return the Ecc data associated with the outbits of the eccGroup

o_eccErrorMask Return a mask for the Ecc data a 1 in the mask means the associated eccData was in error

Return values:

nonzero if unsuccessful

NOTE : This function is ring cache enabled

4.2.3.14 int ecmdClientPerlapi::putSpy (const char * *i_target*, const char * *i_spyName*, const char * *i_data*)

Writes the data buffer into the selected position in the selected spy.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

ECMD_INVALID_SPY Spy name is invalid or Spy is an ECC Grouping

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_SPY_IS_EDIAL Spy is an edial have to use putSpyEnum

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core information of spy to write

i_spyName Name of spy to write to

i_data DataBuffer object that holds data to write into spy

NOTE : This function is ring cache enabled

4.2.3.15 int ecmdClientPerlapi::putSpyEnum (const char * *i_target*, const char * *i_spyName*, const char * *i_enumValue*)

Writes the enum into the selected position in the selected spy.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_SUCCESS if successful

ECMD_INVALID_SPY Spy name is invalid or Spy is an ECC Grouping 2retval ECMD_SPY_NOT_ENUMERATED Spy is not enumerated must use putSpy

ECMD_INVALID_SPY_ENUM if enum value specified is not valid

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position/core information of spy to write

i_spyName Name of spy to write to

i_enumValue String enum value to load into the spy

NOTE : This function is ring cache enabled

4.2.3.16 void ecmdClientPerlapi::ecmdEnableRingCache ()

Enables internal caching of read/writes of scan rings to the chip for functions like getring/getspy/getspr.

Postcondition:

Ring caching is enabled on cache enabled functions

- Functions that support caching are documented in the detailed description of the function
- Functions that do not affect the state of the cache are documented in the detailed description of the function
- Any non-cache enabled function will force a flush of the cache before performing the operation
- Some Dll's may not support ring caching, they will not fail on these functions but you will not see the performance gains

4.2.3.17 int ecmdClientPerlapi::ecmdDisableRingCache ()

Disable internal caching of reads/writes of scan rings.

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

NOTE: A Flush of the cache is performed before disabling the cache

4.2.3.18 int ecmdClientPerlapi::ecmdFlushRingCache ()

Flush all modified data from the internal cache to the hardware, then remove all rings from cache.

Return values:

ECMD_SUCCESS if successful

nonzero if unsuccessful

4.2.3.19 int ecmdClientPerlapi::getArray (const char * *i_target*, const char * *i_arrayName*, const char * *i_address*, char ** *o_data*)

Reads bits from the selected array into the data buffer.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_ARRAY if i_arrayName is not valid for target

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero if unsuccessful

Parameters:

i_target Struct that contains chip and cage/node/slot/position information of array to read

i_arrayName Name of array to read from

o_data DataBuffer object that holds data read from address

i_address Array Address to read from - length of DataBuffer should be set to length of valid address data

See also:

putArray (p. 24) , getArrayMultiple

4.2.3.20 int ecmdClientPerlapi::putArray (const char * *i_target*, const char * *i_arrayName*, const char * *i_address*, const char * *i_data*)

Writes bits from the data buffer into the selected array.

Return values:

ECMD_TARGET_NOT_CONFIGURED if target is not available in the system

ECMD_INVALID_ARRAY if i_arrayName is not valid for target

ECMD_DATA_OVERFLOW Too much data was provided for a write

ECMD_DATA_UNDERFLOW Too little data was provided to a write function

ECMD_SUCCESS if successful

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_CLOCKS_IN_INVALID_STATE Chip Clocks were in an invalid state to perform the operation

nonzero if unsuccessful

ECMD_DATA_OVERFLOW Too much data was provided for a write

Parameters:

i_target Struct that contains chip and cage/node/slot/position information of array to write

i_arrayName Name of array to write to

i_data DataBuffer object that holds data to write into array

i_address Array Address to write to - length of DataBuffer should be set to length of valid address data

See also:
getArray (p. 24)

4.2.3.21 int ecmdClientPerlapi::simaet (const char * *i_function*)

Enable/Disable Simulation AET Logging.

Parameters:
i_function Should be either 'on'/'off'/'flush'

Return values:
ECMD_SUCCESS if successful

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

nonzero on failure

4.2.3.22 int ecmdClientPerlapi::simcheckpoint (const char * *i_checkpoint*)

Store a checkpoint to specified file.

Parameters:
i_checkpoint Name of checkpoint to write to

Return values:
ECMD_SUCCESS if successful

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

nonzero on failure

4.2.3.23 int ecmdClientPerlapi::simclock (int *i_cycles*)

Clock the model.

Parameters:
i_cycles Number of cycles to clock model

Return values:
ECMD_SUCCESS if successful

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

nonzero on failure

4.2.3.24 int ecmdClientPerlapi::simecho (const char * *i_message*)

Echo message to stdout and sim log.

Parameters:
i_message Message to echo

Return values:
ECMD_SUCCESS if successful

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

nonzero on failure

4.2.3.25 int ecmdClientPerlapi::simexit ()

Close down the simulation model.

Return values:
ECMD_SUCCESS if successful

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

nonzero on failure

4.2.3.26 int ecmdClientPerlapi::simEXPECTFAC (const char * *i_facname*, int *i_bitlength*, const char * *i_expect*, int *i_row* = 0, int *i_offset* = 0)

Perform expect on facility using name.

Parameters:
i_facname Facility name

i_expect Value to expect on facility

i_bitlength Length of data to expect

i_row Optional: Array Facility row

i_offset Optional: Facility offset

Return values:
ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

4.2.3.27 int ecmdClientPerlapi::simexpecttcfac (const char * *i_tcfacname*, int *i_bitlength*, const char * *i_expect*, int *i_row* = 0)

Perform expect on TCFAC facility.

Parameters:

i_tcfacname Facility name

i_expect Value to expect on facility

i_bitlength Length of data to expect

i_row Optional: Array Facility row

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

4.2.3.28 int ecmdClientPerlapi::simgetcurrentcycle (char ** *o_cyclecount*)

Fetch current model cycle count.

Parameters:

o_cyclecount Current model cycle count

Return values:

ECMD_SUCCESS if successful

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

nonzero on failure

4.2.3.29 int ecmdClientPerlapi::simGETFAC (const char * *i_facname*, int *i_bitlength*, char ** *o_data*, int *i_row* = 0, int *i_offset* = 0)

Retrieve a Facility using a name.

Parameters:

i_facname Facility name

i_bitlength Bit length to read from facility

o_data Data read from facility

i_row Optional: Array row

i_offset Optional : Facility offset

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

4.2.3.30 int ecmdClientPerlapi::simGETFACX (const char * *i_facname*, int *i_bitlength*, char ** *o_data*, int *i_row* = 0, int *i_offset* = 0)

Retrieve a Facility using a name - preserving Xstate.

Parameters:

i_facname Facility name

i_bitlength Bit length to read from facility

o_data Data read from facility

i_row Optional: Array row

i_offset Optional : Facility offset

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

4.2.3.31 int ecmdClientPerlapi::simgettcfac (const char * *i_tcfacname*, char ** *o_data*, int *i_row* = 0, int *i_startbit* = 0, int *i_bitlength* = 0)

Retrieve a TCFAC facility.

Parameters:

i_tcfacname TCFAC name

o_data Value read

i_row Optional: Array Facility row

i_startbit Optional: Startbit to read

i_bitlength Optional: Length of data to read

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

4.2.3.32 int ecmdClientPerlapi::siminit (const char * *i_checkpoint*)

Initialize the simulation.

Parameters:

i_checkpoint Checkpoint to load : 'none' to skip

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

4.2.3.33 int ecmdClientPerlapi::simPUTFAC (const char * *i_facname*, int *i_bitlength*, const char * *i_data*, int *i_row* = 0, int *i_offset* = 0)

Write a Facility using a name.

Parameters:

i_facname Facility name

i_bitlength Bit length to write to facility

i_data Data to write

i_row Optional: Array row

i_offset Optional : Facility offset

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

4.2.3.34 int ecmdClientPerlapi::simPUTFACX (const char * *i_facname*, int *i_bitlength*, const char * *i_data*, int *i_row* = 0, int *i_offset* = 0)

Write a Facility using a name - preserving Xstate.

Parameters:

i_facname Facility name

i_bitlength Bit length to write to facility

i_data Data to write

i_row Optional: Array row

i_offset Optional : Facility offset

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

4.2.3.35 int ecmdClientPerlapi::simputtcfac (const char * *i_tcfacname*, int *i_bitlength*, const char * *i_data*, int *i_row* = 0, int *i_numrows* = 0)

Write a TCFAC facility.

Parameters:

i_tcfacname TCFAC name

i_data Value to write

i_row Optional: Array Facility row

i_numrows Optional: Number of rows to write

i_bitlength Bit length to write to facility

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

4.2.3.36 int ecmdClientPerlapi::simrestart (const char * *i_checkpoint*)

Load a checkpoint into model.

Parameters:

i_checkpoint Name of checkpoint

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

4.2.3.37 int ecmdClientPerlapi::simSTKFAC (const char * *i_facname*, int *i_bitlength*, const char * *i_data*, int *i_row* = 0, int *i_offset* = 0)

Stick a Facility using a name.

Parameters:

i_facname Facility name

i_bitlength Bit length to stick to facility

i_data Data to stick

i_row Optional: Array row

i_offset Optional : Facility offset

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

4.2.3.38 int ecmdClientPerlapi::simstktcfac (const char * *i_tcfacname*, int *i_bitlength*, const char * *i_data*, int *i_row* = 0, int *i_numrows* = 0)

Stick a TCFAC facility.

Parameters:

i_tcfacname TCFAC name

i_data Value to stick

i_row Optional: Array Facility row

i_numrows Optional: Number of rows to stick

i_bitlength Bit length to write to facility

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

4.2.3.39 int ecmdClientPerlapi::simSUBCMD (const char * *i_command*)

Run RTX SUBCMD.

Parameters:

i_command Command

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

4.2.3.40 int ecmdClientPerlapi::simtckinterval (int *i_tckinterval*)

Set TCK Interval setting in the model for JTAG Master.

Parameters:

i_tckinterval new setting for tck interval when using JTAG

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

4.2.3.41 int ecmdClientPerlapi::simUNSTICK (const char * *i_facname*, int *i_bitlength*, int *i_row* = 0, int *i_offset* = 0)

Unstick a Facility using a name.

Parameters:

i_facname Facility name

i_bitlength Bit length to unstick to facility

i_row Optional: Array row

i_offset Optional : Facility offset

Return values:

ECMD_SUCCESS if successful

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

nonzero on failure

4.2.3.42 int ecmdClientPerlapi::simunsticktcfac (const char * *i_tcfacname*, int *i_bitlength*, const char * *i_data*, int *i_row* = 0, int *i_numrows* = 0)

Unstick a TCFAC facility.

Parameters:

i_tcfacname TCFAC name

i_data Value to unstick to

i_row Optional: Array Facility row

i_numrows Optional: Number of rows to unstick

i_bitlength Bit length to unstick to facility

Return values:

ECMD_RING_CACHE_ENABLED Ring Cache enabled function - must be disabled to use this function

ECMD_SUCCESS if successful

nonzero on failure

4.2.3.43 char* ecmdClientPerlapi::ecmdGetErrorMsg (int *i_errorCode*)

Retrieve additional error information for errorcode.

Parameters:

i_errorCode Error code to lookup up message for

Return values:

point to NULL terminated string containing error data, NULL if error occurs

4.2.3.44 void ecmdClientPerlapi::ecmdOutputError (const char * *i_message*)

Output a message related to an error.

Parameters:

i_message String to output

4.2.3.45 void ecmdClientPerlapi::ecmdOutputWarning (const char * *i_message*)

Output a message related to an warning.

Parameters:
i_message String to output

4.2.3.46 void ecmdClientPerlapi::ecmdOutput (const char * *i_message*)

Output a message to the screen or logs.

Parameters:
i_message String to output

The documentation for this class was generated from the following file:

- **ecmdClientPerlapi.H**

4.3 ecmdDataBuffer Class Reference

Perl Class to provide a means to handle data from the eCMD Perl API.

`#include <ecmdDataBuffer.H>`

Public Methods

- int **getWordLength** () const
  Return the length of the buffer in words.
- int **getByteLength** () const
  Return the length of the buffer in bytes.
- int **getBitLength** () const
  Return the length of the buffer in bits.
- void **setWordLength** (int i_newNumWords)
  Reinitialize the Buffer to specified length.
- void **setBitLength** (int i_newNumBits)
  Reinitialize the Buffer to specified length.
- void **setBit** (int i_bit, int i_len)
  Turn on a bit in buffer.
- void **clearBit** (int i_bit, int i_len)
  Clear a bit in buffer.
- void **setWord** (int i_wordoffset, uint32_t i_value)
  Set a word of data in buffer.
- uint32_t **getWord** (int i_wordoffset)
  Fetch a word from ecmdDataBuffer.
- void **setByte** (int i_byteoffset, uint32_t i_value)
  Set a byte of data in buffer.
- uint8_t **getByte** (int i_byteoffset)
  Fetch a byte from ecmdDataBuffer.
- void **flipBit** (int i_bit, int i_len)
  Invert bit.
- int **isBitSet** (int i_bit, int i_len)
  Test if bit is set.
- int **isBitClear** (int i_bit, int i_len)
  Test if bit is clear.

- int **getNumBitsSet** (int i_bit, int i_len)
  Count number of bits set in a range.
- void **shiftRight** (int i_shiftnum)
  Shift data to right.
- void **shiftLeft** (int i_shiftnum)
  Shift data to left.
- void **rotateRight** (int i_rotatenum)
  Rotate data to right.
- void **rotateLeft** (int i_rotatenum)
  Rotate data to left.
- void **flushTo0** ()
  Clear entire buffer to 0's.
- void **flushTo1** ()
  Set entire buffer to 1's.
- void **invert** ()
  Invert entire buffer.
- void **applyInversionMask** (uint32_t *i_invMask, int i_invByteLen)
  Apply an inversion mask to data inside buffer.
- void **insert** (ecmdDataBuffer &i_bufferIn, int i_start, int i_len)
  Insert part of another DataBuffer into this one.
- void **insertFromRight** (uint32_t *i_datain, int i_start, int i_len)
  Insert a right aligned (decimal) uint32_t array into this DataBuffer.
- void **extract** (ecmdDataBuffer &o_bufferOut, int i_start, int i_len)
  Copy data from this DataBuffer into another.
- void **setOr** (ecmdDataBuffer &i_bufferIn, int i_startbit, int i_len)
  OR data into DataBuffer.
- void **merge** (ecmdDataBuffer &i_bufferIn)
  OR data into DataBuffer.
- void **setAnd** (ecmdDataBuffer &i_bufferIn, int i_startbit, int i_len)
  AND data into DataBuffer.
- void **copy** (ecmdDataBuffer &o_copyBuffer)
  Copy entire contents of this ecmdDataBuffer into o_copyBuffer.
- void **memCopyIn** (uint32_t *i_buf, int i_bytes)
  Copy buffer into this ecmdDataBuffer.

- void **memCopyOut** (uint32_t *o_buf, int i_bytes)

  Copy DataBuffer into supplied uint32_t buffer.

- int **oddParity** (int i_start, int i_stop, int i_insertpos)

  Generate odd parity over a range of bits and insert into DataBuffer.

- int **oddParity** (int i_start, int i_stop, int i_insertpos)

  Generate even parity over a range of bits and insert into DataBuffer.

- std::string **genHexLeftStr** (int i_start, int i_bitlen)

  Return Data as a hex left aligned char string.

- std::string **genHexRightStr** (int i_start, int i_bitlen)

  Return Data as a hex right aligned char string.

- std::string **genBinStr** (int i_start, int i_bitlen)

  Return Data as a binary char string.

- std::string **genXstateStr** (int i_start, int i_bitlen)

  Retrieve a section of the Xstate Data.

- int **insertFromHexLeft** (const char *i_hexChars, int i_start=0, int i_length=0)

  Convert data from a hex left-aligned string and insert it into this data buffer.

- int **insertFromHexRight** (const char *i_hexChars, int i_start=0, int i_expected-Length=0)

  Convert data from a hex right-aligned string and insert it into this data buffer.

- int **insertFromBin** (const char *i_binChars, int i_start=0)

  Convert data from a binary string and insert it into this data buffer.

- int **hasXstate** (int i_start, int i_length)

  Check section of buffer for any X-state values.

- char **getXstate** (int i_bit)

  Retrieve an Xstate value from the buffer.

- void **setXstate** (int i_bit, char i_value)

  Set an Xstate value in the buffer.

- void **memCopyInXstate** (char *i_buf, int i_bytes)

  Copy buffer into the Xstate data of this ecmdDataBuffer.

- void **memCopyOutXstate** (char *o_buf, int i_bytes)

  Copy DataBuffer into supplied char buffer from Xstate data.

4.3.1 Detailed Description

Perl Class to provide a means to handle data from the eCMD Perl API.

Usage :

```
require ecmdDataBuffer;
my $data = new ecmdDataBuffer();
$target->setWord(2,0xFEEDBEEF);
```

4.3.2 Member Function Documentation

4.3.2.1 int ecmdDataBuffer::getWordLength () const

Return the length of the buffer in words.

Return values:
Buffer length in words rounded up

4.3.2.2 int ecmdDataBuffer::getByteLength () const

Return the length of the buffer in bytes.

Return values:
Buffer length in bytes rounded up

4.3.2.3 int ecmdDataBuffer::getBitLength () const

Return the length of the buffer in bits.

Return values:
Buffer length in bits

4.3.2.4 void ecmdDataBuffer::setWordLength (int *i_newNumWords*)

Reinitialize the Buffer to specified length.

Parameters:
i_newNumWords Length of new buffer in words

Postcondition:
Buffer is reinitialized

CAUTION : All data stored in buffer will be lost

4.3.2.5 void ecmdDataBuffer::setBitLength (int *i_newNumBits*)

Reinitialize the Buffer to specified length.

Parameters:
i_newNumBits Length of new buffer in bits

Postcondition:
Buffer is reinitialized

CAUTION : All data stored in buffer will be lost

4.3.2.6 void ecmdDataBuffer::setBit (int *i_bit*, int *i_len*)

Turn on a bit in buffer.

Parameters:
i_bit Bit in buffer to turn on

i_len Number of consecutive bits from start bit to turn on

4.3.2.7 void ecmdDataBuffer::clearBit (int *i_bit*, int *i_len*)

Clear a bit in buffer.

Parameters:
i_bit Bit in buffer to turn off

i_len Number of consecutive bits from start bit to off

4.3.2.8 void ecmdDataBuffer::setWord (int *i_wordoffset*, uint32_t *i_value*)

Set a word of data in buffer.

Parameters:
i_wordoffset Offset of word to set

i_value 32 bits of data to put into word

4.3.2.9 uint32_t ecmdDataBuffer::getWord (int *i_wordoffset*)

Fetch a word from ecmdDataBuffer.

Parameters:
i_wordoffset Offset of word to fetch

Return values:
Value of word requested

4.3.2.10 void ecmdDataBuffer::setByte (int *i_byteoffset*, uint32_t *i_value*)

Set a byte of data in buffer.

Parameters:
i_byteoffset Offset of byte to set
i_value 8 bits of data to put into byte

4.3.2.11 uint8_t ecmdDataBuffer::getByte (int *i_byteoffset*)

Fetch a byte from ecmdDataBuffer.

Parameters:
i_byteoffset Offset of byte to fetch

Return values:
Value of byte requested

4.3.2.12 void ecmdDataBuffer::flipBit (int *i_bit*, int *i_len*)

Invert bit.

Parameters:
i_bit Bit in buffer to invert
i_len Number of consecutive bits to invert

4.3.2.13 int ecmdDataBuffer::isBitSet (int *i_bit*, int *i_len*)

Test if bit is set.

Parameters:
i_bit Bit to test
i_len Number of consecutive bits to test

Return values:
true if bit is set - false if bit is clear

4.3.2.14 int ecmdDataBuffer::isBitClear (int *i_bit*, int *i_len*)

Test if bit is clear.

Parameters:
i_bit Bit to test
i_len Number of consecutive bits to test

Return values:
true if bit is clear - false if bit is set

4.3.2.15 int ecmdDataBuffer::getNumBitsSet (int *i_bit*, int *i_len*)

Count number of bits set in a range.

Parameters:
i_bit Start bit to test

i_len Number of consecutive bits to test

Return values:
Number of bits set in range

4.3.2.16 void ecmdDataBuffer::shiftRight (int *i_shiftnum*)

Shift data to right.

Parameters:
i_shiftnum Number of bits to shift

Postcondition:
Bits in buffer are shifted to right by specified number of bits - data is shifted off the end
Buffer size is unchanged

4.3.2.17 void ecmdDataBuffer::shiftLeft (int *i_shiftnum*)

Shift data to left.

Parameters:
i_shiftnum Number of bits to shift

Postcondition:
Bits in buffer are shifted to left by specified number of bits - data is shifted off the beginning
Buffer size is unchanged

4.3.2.18 void ecmdDataBuffer::rotateRight (int *i_rotatenum*)

Rotate data to right.

Parameters:
i_rotatenum Number of bits to rotate

Postcondition:
Bits in buffer are rotated to the right by specified number of bits - data is rotated to the beginning

4.3.2.19 void ecmdDataBuffer::rotateLeft (int *i_rotatenum*)

Rotate data to left.

Parameters:
i_rotatenum Number of bits to rotate

Postcondition:
Bits in buffer are rotated to the left by specified number of bits - data is rotated to the end

4.3.2.20 void ecmdDataBuffer::flushTo0 ()

Clear entire buffer to 0's.

4.3.2.21 void ecmdDataBuffer::flushTo1 ()

Set entire buffer to 1's.

4.3.2.22 void ecmdDataBuffer::invert ()

Invert entire buffer.

4.3.2.23 void ecmdDataBuffer::applyInversionMask (uint32_t * *i_invMask*, int *i_invByteLen*)

Apply an inversion mask to data inside buffer.

Parameters:
i_invMask Buffer that stores inversion mask

i_invByteLen Buffer length provided in bytes

4.3.2.24 void ecmdDataBuffer::insert (ecmdDataBuffer & *i_bufferIn*, int *i_start*, int *i_len*)

Insert part of another DataBuffer into this one.

Parameters:
i_bufferIn DataBuffer to copy data from - data is taken left aligned

i_start Start bit to insert to

i_len Length of bits to insert

Postcondition:
Data is copied from bufferIn to this DataBuffer in specified location

4.3.2.25 void ecmdDataBuffer::insertFromRight (uint32_t * *i_datain*, int *i_start*, int *i_len*)

Insert a right aligned (decimal) uint32_t array into this DataBuffer.

Parameters:

i_datain uint32_t array to copy into this DataBuffer - data is taken right aligned

i_start Start bit to insert into

i_len Length of bits to insert

Postcondition:

Data is copied from datain into this DataBuffer at specified location

4.3.2.26 void ecmdDataBuffer::extract (ecmdDataBuffer & *o_bufferOut*, int *i_start*, int *i_len*)

Copy data from this DataBuffer into another.

Parameters:

o_bufferOut DataBuffer to copy into - data is placed left aligned

i_start Start bit of data in this DataBuffer to copy

i_len Length of consecutive bits to copy

Postcondition:

Data is copied from specified location in this DataBuffer to bufferOut

4.3.2.27 void ecmdDataBuffer::setOr (ecmdDataBuffer & *i_bufferIn*, int *i_startbit*, int *i_len*)

OR data into DataBuffer.

Parameters:

i_bufferIn DataBuffer to OR data from - data is taken left aligned

i_startbit Start bit to OR to

i_len Length of bits to OR

Postcondition:

Data is ORed from i_bufferIn to this DataBuffer in specified location

4.3.2.28 void ecmdDataBuffer::merge (ecmdDataBuffer & *i_bufferIn*)

OR data into DataBuffer.

Parameters:

i_bufferIn DataBuffer to OR data from - data is taken left aligned

Postcondition:

Entire data is ORed from bufferIn to this DataBuffer

4.3.2.29 void ecmdDataBuffer::setAnd (ecmdDataBuffer & *i_bufferIn*, int *i_startbit*, int *i_len*)

AND data into DataBuffer.

Parameters:

i_bufferIn Bitvector to AND data from - data is taken left aligned

i_startbit Start bit to AND to

i_len Length of bits to AND

Postcondition:

Data is ANDed from bufferIn to this DataBuffer in specified location

4.3.2.30 void ecmdDataBuffer::copy (ecmdDataBuffer & *o_copyBuffer*)

Copy entire contents of this ecmdDataBuffer into o_copyBuffer.

Parameters:

o_copyBuffer DataBuffer to copy data into

Postcondition:

copyBuffer is an exact duplicate of this DataBuffer

4.3.2.31 void ecmdDataBuffer::memCopyIn (uint32_t * *i_buf*, int *i_bytes*)

Copy buffer into this ecmdDataBuffer.

Parameters:

i_buf Buffer to copy from

i_bytes Byte length to copy

Postcondition:

Xstate and Raw buffer are set to value in i_buf for smaller of i_bytes or buffer capacity

4.3.2.32 void ecmdDataBuffer::memCopyOut (uint32_t * *o_buf*, int *i_bytes*)

Copy DataBuffer into supplied uint32_t buffer.

Parameters:

o_buf Buffer to copy into - must be pre-allocated

i_bytes Byte length to copy

Postcondition:

o_buf has contents of databuffer for smaller of i_bytes or buffer capacity

4.3.2.33 int ecmdDataBuffer::oddParity (int *i_start*, int *i_stop*, int *i_insertpos*)

Generate odd parity over a range of bits and insert into DataBuffer.

Parameters:

i_start Start bit of range

i_stop Stop bit of range

i_insertpos Bit position to insert parity

Return values:

0 on success - nonzero on failure

4.3.2.34 int ecmdDataBuffer::oddParity (int *i_start*, int *i_stop*, int *i_insertpos*)

Generate even parity over a range of bits and insert into DataBuffer.

Parameters:

i_start Start bit of range

i_stop Stop bit of range

i_insertpos Bit position to insert parity

Return values:

0 on success - nonzero on failure

4.3.2.35 std::string ecmdDataBuffer::genHexLeftStr (int *i_start*, int *i_bitlen*)

Return Data as a hex left aligned char string.

Parameters:

i_start Start bit of data to convert

i_bitlen Number of consecutive bits to convert

Return values:

String containing requested data

4.3.2.36 std::string ecmdDataBuffer::genHexRightStr (int *i_start*, int *i_bitlen*)

Return Data as a hex right aligned char string.

Parameters:

i_start Start bit of data to convert

i_bitlen Number of consecutive bits to convert

Return values:

String containing requested data

4.3.2.37 std::string ecmdDataBuffer::genBinStr (int *i_start*, int *i_bitlen*)

Return Data as a binary char string.

Parameters:

i_start Start bit of data to convert

i_bitlen Number of consecutive bits to convert

Return values:

String containing requested data

4.3.2.38 std::string ecmdDataBuffer::genXstateStr (int *i_start*, int *i_bitlen*)

Retrieve a section of the Xstate Data.

Parameters:

i_start Start bit of data to retrieve

i_bitlen Number of consecutive bits to retrieve

Return values:

String containing requested data

4.3.2.39 int ecmdDataBuffer::insertFromHexLeft (const char * *i_hexChars*, int *i_start* = 0, int *i_length* = 0)

Convert data from a hex left-aligned string and insert it into this data buffer.

Parameters:

i_hexChars Hex Left-aligned string of data to insert

i_start Starting position in data buffer to insert to, 0 by default

i_length Length of data to insert, defaults to length of i_hexChars, zeroes are padded or data dropped from right if necessary

Return values:

ECMD_DBUF_INVALID_DATA_FORMAT if non-hex chars detected in i_hexChars

ECMD_SUCCESS on success

non-zero on failure

4.3.2.40 int ecmdDataBuffer::insertFromHexRight (const char * *i_hexChars*, int *i_start* = 0, int *i_expectedLength* = 0)

Convert data from a hex right-aligned string and insert it into this data buffer.

Parameters:

i_hexChars Hex Right-aligned string of data to insert

i_expectedLength The expected length of the string data, zeros are padded or data dropped from the left if necessary

i_start Starting position in data buffer to insert to, 0 by default

Return values:

ECMD_DBUF_INVALID_DATA_FORMAT if non-hex chars detected in i_hexChars

ECMD_SUCCESS on success

non-zero on failure

4.3.2.41 int ecmdDataBuffer::insertFromBin (const char * *i_binChars*, int *i_start* = 0)

Convert data from a binary string and insert it into this data buffer.

Return values:

0 on success- non-zero on failure

Parameters:

i_binChars String of 0's and 1's to insert

i_start Starting position in data buffer to insert to, 0 by default

Return values:

ECMD_DBUF_INVALID_DATA_FORMAT if non-binary chars detected in i_binChars

ECMD_SUCCESS on success

non-zero on failure

4.3.2.42 int ecmdDataBuffer::hasXstate (int *i_start*, int *i_length*)

Check section of buffer for any X-state values.

Parameters:

i_start Start bit to test

i_length Number of consecutive bits to test

Return values:

1 if xstate found 0 if none

4.3.2.43 char ecmdDataBuffer::getXstate (int *i_bit*)

Retrieve an Xstate value from the buffer.

Parameters:

i_bit Bit to retrieve

NOTE - To retrieve multiple bits use genXstateStr

4.3.2.44 void ecmdDataBuffer::setXstate (int *i_bit*, char *i_value*)

Set an Xstate value in the buffer.

Parameters:

i_bit Bit to set

i_value Xstate value to set

4.3.2.45 void ecmdDataBuffer::memCopyInXstate (char * *i_buf*, int *i_bytes*)

Copy buffer into the Xstate data of this ecmdDataBuffer.

Parameters:

i_buf Buffer to copy from

i_bytes Byte length to copy (char length)

Postcondition:

Xstate and Raw buffer are set to value in i_buf for smaller of i_bytes or buffer capacity

4.3.2.46 void ecmdDataBuffer::memCopyOutXstate (char * *o_buf*, int *i_bytes*)

Copy DataBuffer into supplied char buffer from Xstate data.

Parameters:

o_buf Buffer to copy into - must be pre-allocated

i_bytes Byte length to copy (char length)

Postcondition:

o_buf has contents of databuffer for smaller of i_bytes or buffer capacity

The documentation for this class was generated from the following file:

- **ecmdDataBuffer.H**

Chapter 5

eCMD Perl Module File Documentation

5.1 ChipTarget.H File Reference

Perl Class to hold eCMD Targetting information.

Compounds

- class **ChipTarget**
  Perl Class to hold eCMD Targetting information.

5.1.1 Detailed Description

Perl Class to hold eCMD Targetting information.

Usage :

```
require ChipTarget (p. 9);
my $target = new ChipTarget (p. 9)("pu -p1");
$target->node(2);
```

5.2 ecmdClientPerlapi.H File Reference

eCMD Perl API.

```
#include <string>
```

```
#include <ecmdStructs.H>
```

Compounds

- class **ecmdClientPerlapi**
  eCMD Perl API.

Functions

- int **ecmdPerlInterfaceErrorCheck** (int ec)

5.2.1 Detailed Description

eCMD Perl API.

Usage :

```
@filestat = stat("/usr/include/linux/zorro.h");
if (@filestat[1] eq "") {
  require ecmd_aix_perl;
} else {
  require ecmd_x86_perl;
}
my $ep = new ecmdClientPerlapi();
```

5.2.2 Function Documentation

5.2.2.1 int ecmdPerlInterfaceErrorCheck (int *ec*)

5.3 ecmdDataBuffer.H File Reference

Provides a means to handle data from the eCMD Perl API.

Compounds

- class **ecmdDataBuffer**

  Perl Class to provide a means to handle data from the eCMD Perl API.

5.3.1 Detailed Description

Provides a means to handle data from the eCMD Perl API.

DataBuffers handle data as a binary string

Usage :

```
require ecmdDataBuffer (p. 34);
my $data = new ecmdDataBuffer();
$target->setWord(2,0xFEEDBEEF);
```

Index