```
* For the user
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

NOTE: This document describes the usage of the high level CI API as
in accordance to the Linux DVB API. This is a not a documentation for the,
existing low level CI API.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


To utilize the High Level CI capabilities,

(1*) This point is valid only for the Twinhan/clones
  For the Twinhan/Twinhan clones, the dst_ca module handles the CI
  hardware handling.This module is loaded automatically if a CI
  (Common Interface, that holds the CAM (Conditional Access Module)
  is detected.

(2) one requires a userspace application, ca_zap. This small userland
  application is in charge of sending the descrambling related information
  to the CAM.

This application requires the following to function properly as of now.

        (a) Tune to a valid channel, with szap.
          eg: $ szap -c channels.conf -r "TMC" -x

        (b) a channels.conf containing a valid PMT PID
          eg: TMC:11996:h:0:27500:278:512:650:321

          here 278 is a valid PMT PID. the rest of the values are the
          same ones that szap uses.

        (c) after running a szap, you have to run ca_zap, for the
          descrambler to function,
          eg: $ ca_zap channels.conf "TMC"

        (d) Hopefully enjoy your favourite subscribed channel as you do with
          a FTA card.

(3) Currently ca_zap, and dst_test, both are meant for demonstration
  purposes only, they can become full fledged applications if necessary.


* Cards that fall in this category
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

At present the cards that fall in this category are the Twinhan and its
clones, these cards are available as VVMER, Tomato, Hercules, Orange and
so on.

* CI modules that are supported
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

The CI module support is largely dependant upon the firmware on the cards
Some cards do support almost all of the available CI modules. There is
nothing much that can be done in order to make additional CI modules
working with these cards.

Modules that have been tested by this driver at present are
```

(1) Irdeto 1 and 2 from SCM
(2) Viaccess from SCM
(3) Dragoncam

* The High level CI API

* For the programmer

With the High Level CI approach any new card with almost any random architecture can be implemented with this style, the definitions inside the switch statement can be easily adapted for any card, thereby eliminating the need for any additional ioctls.

The disadvantage is that the driver/hardware has to manage the rest. For the application programmer it would be as simple as sending/receiving an array to/from the CI ioctls as defined in the Linux DVB API. No changes have been made in the API to accommodate this feature.

* Why the need for another CI interface ?

This is one of the most commonly asked question. Well a nice question. Strictly speaking this is not a new interface.

The CI interface is defined in the DVB API in ca.h as

```
typedef struct ca_slot_info {
        int num;               /* slot number */

        int type;              /* CA interface this slot supports */
#define CA_CI            1     /* CI high level interface */
#define CA_CI_LINK       2     /* CI link layer level interface */
#define CA_CI_PHYS       4     /* CI physical layer level interface */
#define CA_DESCR         8     /* built-in descrambler */
#define CA_SC          128     /* simple smart card interface */

        unsigned int flags;
#define CA_CI_MODULE_PRESENT 1 /* module (or card) inserted */
#define CA_CI_MODULE_READY   2
} ca_slot_info_t;
```

This CI interface follows the CI high level interface, which is not implemented by most applications. Hence this area is revisited.

This CI interface is quite different in the case that it tries to accommodate all other CI based devices, that fall into the other categories.

This means that this CI interface handles the EN50221 style tags in the Application layer only and no session management is taken care of by the application. The driver/hardware will take care of all that.

This interface is purely an EN50221 interface exchanging APDU's. This means that no session management, link layer or a transport layer do

exist in this case in the application to driver communication. It is
as simple as that. The driver/hardware has to take care of that.

With this High Level CI interface, the interface can be defined with the
regular ioctls.

All these ioctls are also valid for the High level CI interface

```
#define CA_RESET          _IO('o', 128)
#define CA_GET_CAP        _IOR('o', 129, ca_caps_t)
#define CA_GET_SLOT_INFO  _IOR('o', 130, ca_slot_info_t)
#define CA_GET_DESCR_INFO _IOR('o', 131, ca_descr_info_t)
#define CA_GET_MSG        _IOR('o', 132, ca_msg_t)
#define CA_SEND_MSG       _IOW('o', 133, ca_msg_t)
#define CA_SET_DESCR      _IOW('o', 134, ca_descr_t)
#define CA_SET_PID        _IOW('o', 135, ca_pid_t)
```

On querying the device, the device yields information thus

```
CA_GET_SLOT_INFO
----------------------------
Command = [info]
APP: Number=[1]
APP: Type=[1]
APP: flags=[1]
APP: CI High level interface
APP: CA/CI Module Present

CA_GET_CAP
----------------------------
Command = [caps]
APP: Slots=[1]
APP: Type=[1]
APP: Descrambler keys=[16]
APP: Type=[1]

CA_SEND_MSG
----------------------------
Descriptors(Program Level)=[ 09 06 06 04 05 50 ff f1]
Found CA descriptor @ program level

(20) ES type=[2] ES pid=[201]  ES length =[0 (0x0)]
(25) ES type=[4] ES pid=[301]  ES length =[0 (0x0)]
ca_message length is 25 (0x19) bytes
EN50221 CA MSG=[ 9f 80 32 19 03 01 2d d1 f0 08 01 09 06 06 04 05 50 ff f1 02 e0
c9 00 00 04 e1 2d 00 00]
```

Not all ioctl's are implemented in the driver from the API, the other
features of the hardware that cannot be implemented by the API are achieved
using the CA_GET_MSG and CA_SEND_MSG ioctls. An EN50221 style wrapper is
used to exchange the data to maintain compatibility with other hardware.

```
/* a message to/from a CI-CAM */
typedef struct ca_msg {
        unsigned int index;
        unsigned int type;
        unsigned int length;
        unsigned char msg[256];
} ca_msg_t;
```

The flow of data can be described thus,

```
        App (User)
        -----
        parse
          |
          |
          v
        en50221 APDU (package)
   --------------------------------------
   |      |                              | High Level CI driver
   |      |                              |
   |      v                              |
   |    en50221 APDU (unpackage)         |
   |      |                              |
   |      |                              |
   |      v                              |
   |    sanity checks                    |
   |      |                              |
   |      |                              |
   |      v                              |
   |    do (H/W dep)                     |
   --------------------------------------
          |    Hardware
          |
          v
```

The High Level CI interface uses the EN50221 DVB standard, following a standard ensures futureproofness.