# Advanced testing for highly dynamic CANopen systems such as CiA 447

Olaf Pfeiffer, Embedded Systems Academy GmbH, Chairman SiG CiA 447 car add-on devices

**Highly dynamic systems supporting plug-and-play require more advanced testing methods both at the device level as well as at the integration level. This paper summarizes the tests available and used for CiA 447[4]. These are the CiA 447 version of the CANopen conformance test, an enhanced "Concise Device Configuration File" format, the CANopen Test Machine and an automated system event analyzer for the integration phase.**
**The "Concise Device Configuration File" format was conceived to support a simple method to configure a CANopen device. The CDCF is primarily a list of write accesses for an Object Dictionary. A CANopen device with a CANopen SDO client can easily process a CDCF by executing the SDO write accesses one by one. However, there is no interaction or flow control supported. The enhanced CDCF format by Embedded Systems Academy supports various commands including setting timeouts, reading back values for confirmation, transmitting the NMT master message and executing a LSS Master cycle. These commands support custom test sequences as well as complex bootloader control.**
**The CANopen Test Machine was jointly defined by Daimler and Embedded Systems Academy and supports creating and editing test graphs drawn in Microsoft Visio®. Although developed and optimized for the CiA 447 SiG it can be used with any CANopen device. The system supports complex test sequences using multiple timers, buffers and CAN messages. The tests are state diagram based and each transition can be based on a condition (check variable or timer) and contain an action (modifying variable or timer) and a CAN transmit or receive. The generated test files can be executed on various hardware platforms, which generate a log file containing the device ID of the DUT as well as a signature line.**

CANopen systems using dynamic node IDs assigned through Layer Setting Services (LSS) with every power up are challenging to debug as Node ID numbers may change with every power cycle. In case of errors it is not immediately clear which physical device is at fault. In regards to test running on a network with multiple devices this means that at all stages of the test it must be clear "who is who" in the network. This paper summarizes the four test utilities currently in use for testing CiA 447 single devices or complete systems:

- (A) Enhanced cctt (CANopen conformance test tool)
- (B) Extended CDCF (Concise Device Configuration File)
- (C) CANopen Test Machine
- (D) Logxaminer CANopen Event Examiner

## (A) Enhanced cctt

A CiA 447 device does not operate unless a CiA 447 gateway is present to wake up the device, assign it a node ID via LSS and keeping it alive by its presence. Without the gateway, a CiA 447 device would fall back into sleep mode.

CiA 447 device tests with the original CANopen Conformance Test Tool are only possible, if a gateway is present on the network at the same time. However, gateways do not only require additional simulations to operate, they also produce background traffic influencing the tests, like using SDO channels or NMT master messages at the same time as the test does.

Therefore the engineers of Embedded Systems Academy created a modified CiA 447 version of the cctt that has the following changes implemented.

1) Pre and post test sequence: each individual test executes a pre and a post sequence. The pre sequence wakes uo the CiA 447 device and assigns it a node ID via LSS. The post sequence sends the device back into sleep mode.
2) Keep alive function: a gateway heartbeat is produced as background function. This ensures that devices do not go back to sleep by themselves.
3) Extended SDO channels: in CiA 447 each device implements 16 SDO channels (default CiA 301[1] plus 15 CiA 447 defined channels). The CiA 447 cctt version allows selecting the SDO channel used during tests.
4) Hiding not applicable tests: Some cctt tests cannot be applied to CiA 447 devices. These are hidden from the user.

In summary these changes ensure that a CiA 447 device can be tested “stand alone” by the cctt and that the pre-conditions for each individual test are the same.

**(B) Extended CDCF**

The “Concise Device Configuration File” is specified in CiA 302-3 [2]. In short, it is an array of records with the following contents:

- Index (type UNSIGNED16)
- Subindex (type UNSIGNED8)
- Length (type UNSIGNED32)
- Data (type DOMAIN – any length)

A CANopen Master or test utility can process this list and “execute” it as a sequence of SDO (Service Data Object) write accesses to a specific node, the device under test (DUT). In traditional CANopen systems, this file format can be used to configure a CANopen node, for example setting specific heartbeat and/or PDO (Process Data Object) transmission modes or event times. When specific sequences are required, then these sequences must be present in the CDCF (e.g. disable PDO, modify parameters, enable PDO).

**Extended CDCF**

As is, the CDCF is just a list of data for Object Dictionary entries. There is no timing information (how fast to execute the sequence) or device information (to identify if this file is for a specific device only) or flow control (only continue if the CANopen device has specific values at selected Object Dictionary entries).

There are no physical layer settings associated with a CDCF: CAN bitrate used, timeouts and delays for SDO transfers or the node ID of the device this file is intended for.

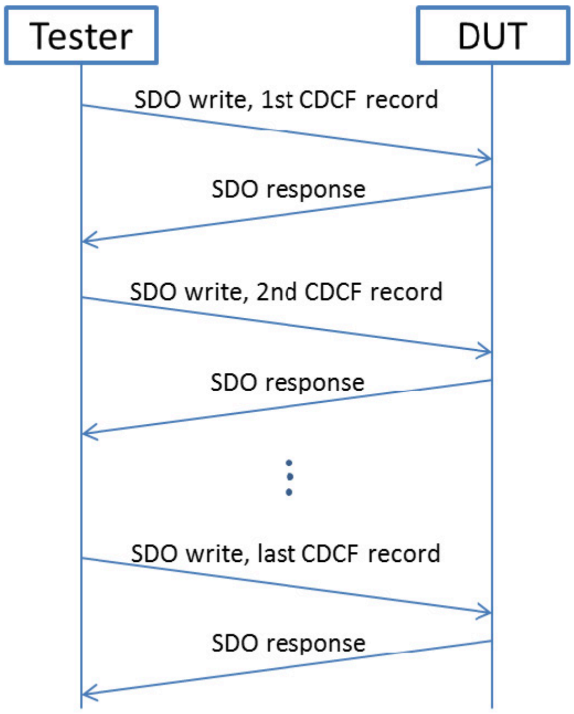
Tester
DUT
SDO write, 1st CDCF record
SDO response
SDO write, 2nd CDCF record
SDO response
SDO write, last CDCF record
SDO response

*Figure1: Traditional CDFC usage*

**Custom sequences**

In order to make the CDCF usable for customizable test sequences or device identification, commands need to be specified. Command extensions like file identification, version and comments allow an identification of a specific CDCF. CDCF player settings support setting of CAN bitrate, node ID, SDO timeouts and delays. Parameters like the number of retries on failures and logging options for debugging and test of the CDCF. Active Controls allow the definition of pauses or a wait for action of the selected device, such as a bootup message or an operational heartbeat. A command can also initiate the execution of a LSS Master Cycle or an NMT Master message. SDO read access with an optional comparison may be executed instead of a write access.

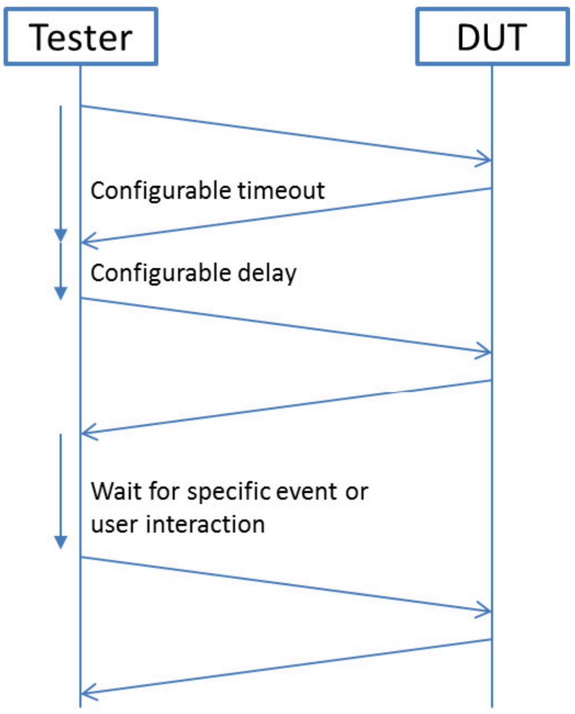

*Figure 2: Timeouts and delays*

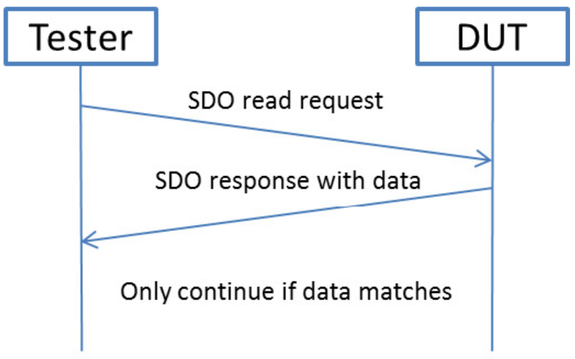

*Figure 3: Match data to identify DUT*

## Command definition

To keep the new CDCF enhancements backward compatible, all new commands are “hidden” in the existing regular records with Object Dictionary entries. The extend commands defined stick to the existing record format. An Index value of 0F0Fh is used to identify enhanced commands. In CANopen this index value is currently reserved [1].

Whenever the device executing the records of a CDCF reaches a record with the Index 0F0Fh, it does not immediately generate an SDO write but interprets the contents as an extended command to execute and processes it accordingly.

A CDCF executing device that is not capable of interpreting these commands will generate an SDO write to Index 0F0Fh which will result in an SDO Abort as 0F0Fh is a reserved value in the CANopen Object Dictionary.

## Extended CDCF commands summary

A detailed definition of the CDCF commands does not fit into the scope of this paper. In short the commands defined are:

1) Informational strings to identify the file and to add conditional outputs to a log file generated, allowing error messages to be added to log files. Strings can also be user requests, like asking the user to push a button at the device before continuing the test.
2) The player settings allow setting bitrate, timeouts and delays as well as retries and logging details.
3) Active control commands include delays/pauses, waiting for an action such as a device bootup message, sending a NMT command or executing a LSS master cycle.
4) SDO accesses may also be reads, reads with a comparison and writes may include writing back a value previously read.

## Usage example: Identification of device

A CDCF can now be associated with a specific device or device class by matching up its Vendor ID, Product Code, Revision information or even serial number. Execution aborts, if the desired entries do not match. For both the test or bootloader sequences this ensures that a CDCF only gets executed on those devices it was generated for. A CDCF can be generated to only match and work with an individual device (match down to serial number) or any device from a series (match to vendor ID, product code and possibly revision number).

## Usage example: Test sequences

Supporting user interactions allows end of production line testing for output devices such as roof bars for police cars as defined by CiA 447[4]. After each switch of an output, a user interaction like “verify if light xyz is now on” can be displayed on the CDCF player. The person running the test can now manually trigger execution of the next sequence by pushing a button/dial on the CDCF player.

## Usage example: Boot loading

A CDCF can now contain all data and commands for boot loading a specific device. First a match verifies that this is the correct device, then the bootloader gets activated and last the sequence is executed to re-program/flash the firmware in the target device. This allows sending an “end user” device specific firmware update files that cannot accidently be programmed into the wrong device as it can be associated to the specific device the user has in his system.

## File generation and editing

Embedded Systems Academy provides a converter utility program that allows generating a CDCF from a comma separated value file as creatable with any spread sheet program such as Microsoft Excel. It simply contains the columns Index, Subindex and Data, the length information gets automatically calculated and inserted. If the data is too big to fit into the data column, the entry can also refer to a file that later gets inserted as data for this entry.

## (C) CANopen test machine

Real-world test examples have shown that timings, timeouts and behavior under stress are major factors for interoperability of CANopen devices. Unfortunately, these are only checked on a very basic level – or not at all – by the CANopen Conformance Test (cct) of the CiA. The CANopen Test Machine provides a framework for customized tests with strict real-time and application-profile-specific requirements.

The CANopen Test Machine allows the creation of tests based on Microsoft Visio® graphs, provided macros then auto-generate a test script from the graph. It is a real-time capable test system intended for the test of CANopen devices. Originally developed for testing CiA 447 application profile compatible devices, it can also be used for CANopen devices implementing any other CANopen device or application profile.

## CANopen test machine basics

The individual tests are based on a state machine with transition rules triggered by the transmission or reception of a CAN message and conditions that include timers and counters. Each individual transition rule can contribute towards a test result by specifying if this transition should add to the fail counter.

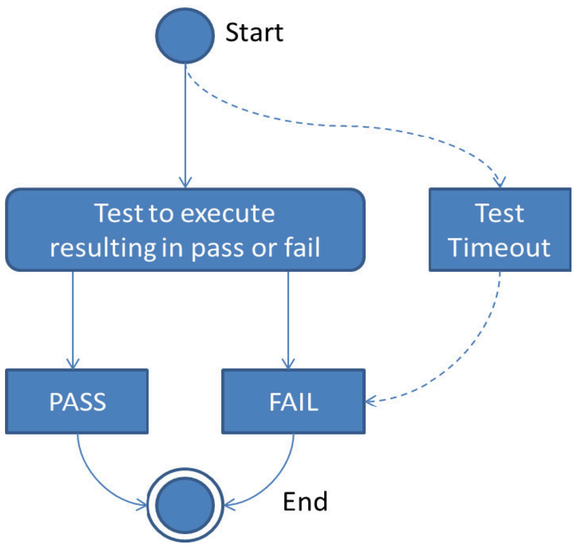

*Figure 4: Test state transitions*

Predefined “smart modes” allow the CAN IDs used in rules to be modified during run-time, for example to include the node ID of the current device-under-test (DUT). Smart modes may also execute autonomous, predefined, complex sequences, such as the LSS Master cycle to detect a node and assign a node ID.

Any single test is automatically monitored with a general test timeout. This ensures that the test always terminates and cannot execute indefinitely. The result of each test must be either a “pass” or a “fail”. If the global timeout has expired before the test terminates regularly, the test is assigned a “failed” result.

For each instance of a test run, a log is generated that lists all applied transition rules. The result is a listing of the steps taken and the result of each step (pass or fail).

The test engine can be executed on various hardware platforms. The main difference between the platforms is the real-time capability that a platform can provide. Some might allow timings generated and verified down to Milliseconds and below, others might only be capable of handling tens of Milliseconds. Tests may specify that a minimal timing resolution is required to execute a test. If the test hardware is not capable of that resolution, it denies execution of the test.

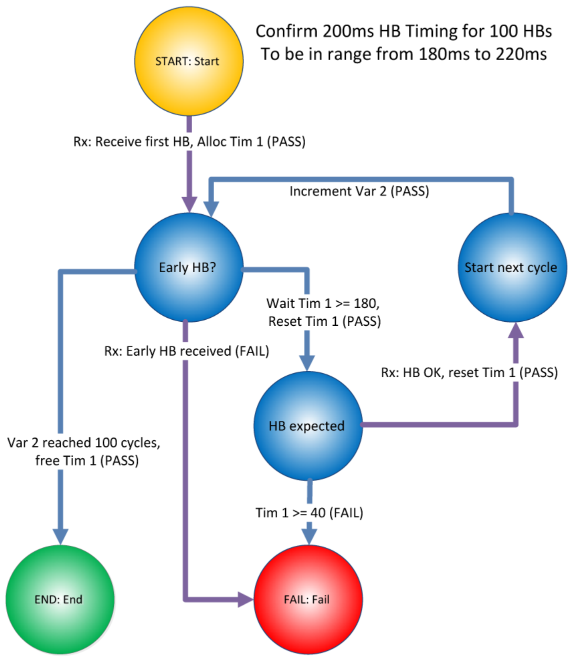
Confirm 200ms HB Timing for 100 HBs
To be in range from 180ms to 220ms
START: Start
Rx: Receive first HB, Alloc Tim 1 (PASS)
Increment Var 2 (PASS)
Early HB?
Start next cycle
Wait Tim 1 >= 180, Reset Tim 1 (PASS)
Rx: Early HB received (FAIL)
Rx: HB OK, reset Tim 1 (PASS)
HB expected
Var 2 reached 100 cycles, free Tim 1 (PASS)
Tim 1 >= 40 (FAIL)
END: End
FAIL: Fail

*Figure 5: Example - heartbeat testing*

The example shown in figure 5 verifies if a 200ms heartbeat timing occurs in a time window from 180ms to 220ms. With each receipt of the heartbeat, timer 1 is restarted. When it reaches 180ms, a 40ms time window opens to receive the next heartbeat. With each receipt variable 2 is incremented, the tests ends after 100 cycles.

### Test Machine timers

All timings and timers are defined in Microseconds. It depends on the platform that executes the test engine which real-time capability is provided. A PC based test engine might be able to monitor the reception of CAN messages with a resolution of a few Microseconds. However, on PC based systems controlled transmission is typically only available with a large jitter (multiple tens of Milliseconds) depending on the performance of the PC and other programs running at the same time. The test engine on the CANopen Diag hardware offers controlled transmissions with a resolution of one Millisecond and reception timestamps with a one Microsecond resolution.

### Test Machine variables

For the flow control of the tests, variables can be used. The data types available for the variables are BOOLEAN, UNSIGNED32 or TIME. The variables are numbered from 1 to the number of variables supported by the hardware executing the tests. The functions to change variables are limited to:

- BOOLEAN: set to TRUE, set to FALSE or TOGGLE
- UNSIGNED32: set to a value, add a value or subtract a value
- TIME: reset and start (as one function) or stop

The condition checks available for each variable are:

- BOOLEAN: is it TRUE or FALSE?
- UNSIGNED32: is content “equal to a value”, is it “>= then a value” or is it “< then a value”?
- TIME: is time expired?

### State transition rules

There are multiple rules defined for state transitions. Every rule may have a precondition check using a single variable or timer and every rule may set/change a variable. This allows executing a rule only when a variable based precondition is met.

Each rule may influence the action to be taken in regards to the test log (pass, fail) and in regards to modifying the timer/counters. For CAN receive rules and time rules, there is also an “any state”. Rules of this virtual state are continuously processed in any state. This allows specifying background messages that could be received in any state or a global timer/counter that is always checked.

## Summary of the CAN transmit rule

- If precondition is met (single variable check) AND current state matches
- Then transmit the specified CAN message
    - Data may come from a buffer
- If a timestamp is specified with the CAN message, then this delay is added BEFORE sending the message
- Add action log for test result
- Perform variable action (influence a single variable)
- Go to next state as defined by rule

## Summary of the CAN receive rule

- If precondition is met (single variable check) and current state matches
- Take next CAN message from receive queue and check for match
    - Match pattern supported to define ranges or “don’t care” values
    - Data received can be copied to a buffer
- Add action log for test result (PASS, FAIL)
- Perform variable action (influence a single variable)
- Go to next state as defined by rule

## Summary of the variable/timer rule

- If precondition is met (single variable check) and current state matches
- Add action log for test result (IDLE, PASS, FAIL)
- Perform variable action (influence a single variable)
- Go to next state as defined by rule

## File conversions

Using macros, the drawn test graphs can be converted into transition rule tables for the CANopen Test Machine player. In addition a pseudo-code script is auto generated listing all the test states with all the possible state transition rules. This ensures double documentation: besides the graph that already documents the test the user also gets a readable test script. The script below belongs to an alternative heartbeat verification test.

```
File generated by ESAcademy‘s CANopen Test Machine Validator
Version 1.00.2754 of 13-JUN-2013

File name: CiA447_HBtime_V102.esat
Full name: CiA447 heartbeat timing
  Version: V01.02 of 31-MAR-2013
DUTnodeID: loop from 2 to 16
Functions: LSS_Master Smart_ID CiA_447_ID use_post_seqence
Resources: 2 timers, 0 buffers, 1000us timer resolution
  Timeout: 15.000s
Descript.: Check if HB between 190-210, do for pre-op,
operational, try with all node IDs

State START:
  ON TRANSMIT {0000h,2,81h,00h,00h,00h,00h,00h,00h,00h}
    LOG: „NMT Master: Reset all nodes“
    GOTO State 1010h

State 1010h: „Wait for boot up of device“
  ON  RECEIVE {0700h, 1,00h,00h,00h,00h,00h,00h,00h,00h}
    AND MATCH {fffh,fh,00h,00h,00h,00h,00h,00h,00h,00h}
              (Smart CAN ID based on DUT)
    LOG: „Node bootup“
    GOTO State 1011h

State 1011h: „Write to OD 1017h,0 of 200“
  ON TRANSMIT {0600h,8,2bh,17h,10h,00h,C8h,00h,00h,00h}
              (Smart CAN ID based on DUT)
    LOG: „Set heartbeat time to 200ms“
    START Timer[01h]
    GOTO State 1012h

State 1012h:
  ON  RECEIVE {0580h, 8,60h,17h,10h,00h,00h,00h,00h,00h}
    AND MATCH {fffh,fh,ffh,ffh,ffh,ffh,ffh,ffh,ffh,ffh}
              (Smart CAN ID based on DUT, client 1)
    LOG: „1017h write response“
    FREE Timer[01h]
    GOTO State 2010h
  ON Timer[01h] >= TIME_MS[50d]
    LOG: „SDO timeout after write 1017h“
    GOTO State FAIL

State 2010h:
  START Timer[01h]
    LOG: „Start new HB timer interval“
    GOTO State 3020h

State 3020h:
  ON Timer[01h] >= TIME_MS[190d]
    LOG: „PASS on no early heartbeat“
    GOTO State 3021h
  ON RxCounter[02h] >= 10
    LOG: „Pass on 10 cycles“
    GOTO State PASS „Completed Test Cycle“

State 3021h:
  START Timer[01h]
    LOG: „time window for heartbeat receive opens“
    GOTO State 3022h

State 3022h:
  ON  RECEIVE {0700h, 1,7fh,00h,00h,00h,00h,00h,00h,00h}
    AND MATCH {fffh,fh,ffh,00h,00h,00h,00h,00h,00h,00h}
              (Smart CAN ID based on DUT)
    LOG: „Next heartbeat received“
    INCrement RxCounter[02h]
    GOTO State 301fh „Continous HB monitoring“
  ON Timer[01h] >= TIME_MS[20d]
    LOG: „FAIL on heartbeat not received in time window“
    GOTO State FAIL

State 301fh: „Continous HB monitoring“
  START Timer[01h]
    LOG: „Start new HB timer interval“
    GOTO State 3020h
```

## Executing tests and logging results

When CANopen Test Machine tests are executed, the CANopen test player generates a log file with all state transitions executed. This log file summarizes if a test passed or at which transitions failures happened. Depending on player settings these can have different level of detail, including optional debug information. The log files have a digital signature in order to protect them from accidental modification.

## CiA 447 gateway testing

In general, tests implemented based on the CANopen Test Machine operate directly on the DUT with no other devices connected. However, in order to test a CiA 447 gateway, an additional tester device as specified in CiA 447 [4] must be present. The test machine test can then send commands to the tester device to produce directed or non-directed background traffic of a specified load.

## (D) Logxaminer event examiner

All tests listed so far are primarily intended for direct use with a single DUT. For final integration where multiple devices are present on the network, additional, non-intrusive methods are required for testing.

The Logxaminer by Embedded Systems Academy is a system event analyzer and implements diagnostics methods used for live monitoring of dynamic systems as well as automated post analysis based on long-term trace recordings. The diagnostic methods described help to quickly pinpoint potential problems in a highly dynamic CANopen system.

The focus of the utility lies on identifying system relevant behavior, such as LSS node ID assignment (including identification so that it is clear “who is who” in current network), bootups, emergencies, monitoring various known sequences and identifying potential duplicate node ID scenarios.

For each live recording or log analysis an “event log” is generated showing these main system events and highlighting potential erroneous behavior.

## Summary and outlook

The test tools addressed in this paper provide advanced testing methods for highly dynamic CANopen systems. Both the extended CDCF format as well as the CANopen Test Machine allow the generation of custom tests also for application profiles defining custom CANopen behavior and functionality.

For the CiA 447 group the next steps are to further define the individual tests in the test plan CiA 312[5]. This test plan is currently under development.

Olaf Pfeiffer
Embedded Systems Academy GmbH
Bahnhofstr. 17
DE-30890 Barsinghausen
Tel.: +49 5105 582 7897
opfeiffer@esacademy.de
www.esacademy.de

## References

[1] CiA 301, CANopen application layer and communication profile
[2] CiA 302-3, Additional application layer functions, configuration and program download
[3] CiA 305, Layer Setting Services (LSS) and protocols
[4] CiA 447, car add-on devices
[5] CiA 312-5, Test plan car add-on devices