Quick Introduction To CAPL

Version 1.1
02/09/03

Application Note AN-AND-1-113

Author(s) Bruce Emaus
Restrictions Public Document
Abstract This application note is a brief introduction to CAPL, the easy-to-use C-based programming language and integrated programming environment inside both CANalyzer and CANoe. The focus is to help the beginning CAPL programmer.

Table of Contents

1.0 CAPL Programming

For CAN-based networks, modules, and distributed embedded systems, Communication Application Programming Language, CAPL, makes it possible to program the CANalyzer for developer-specific applications that use the CAN protocol. CAPL may also be used in the CANoe tool for distributed product development,

2.0 CAPL Applications

2.1 Node Emulation

Using the programmable version of CANalyzer, one can emulate functions of a node including the transmission of event, periodic, and conditionally periodic messages. Users can create conversational responding messages to the reception of designated messages.

2.2 Network Emulation

Using CAPL, CANalyzer can emulate the system-level data traffic of all remaining nodes.

Contact Information: www.vector-cantech.com or 1-248-449-9290

2.3 Node Testing

During a portion of CAN-based module development, CANalyzer can be used to test the module's communication behavior. Evaluation of message timing, handling of Bus Off, and other functions are easy to accomplish.

2.4 Gateway

The programmable CANalyzer can be used as a temporary or permanent gateway (or bridge) between different CAN buses to exchange data.

2.5 Bus Separator

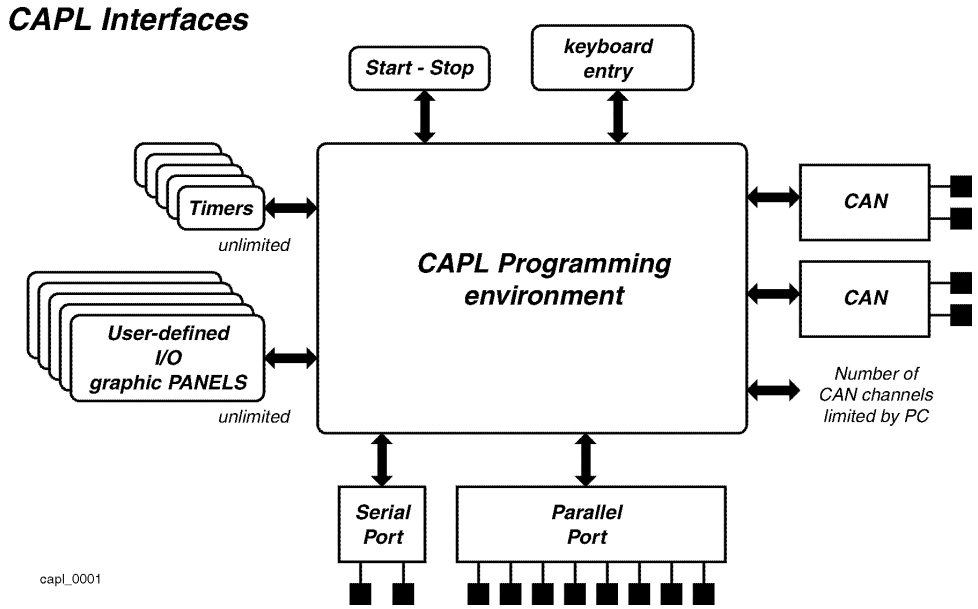
Another target application is to insert CANalyzer Pro between a node to be tested and the actual network. Such configurations can be used during system-level development to isolate (and optionally modify) behavior. The ability to interconnect modules at different revision levels to the distributed product in order to continue development is also another possible application.

3.0 CAPL Programming Environment

Even though CAPL can be thought of as a procedural programming language, CAPL can also be considered as a programming environment.

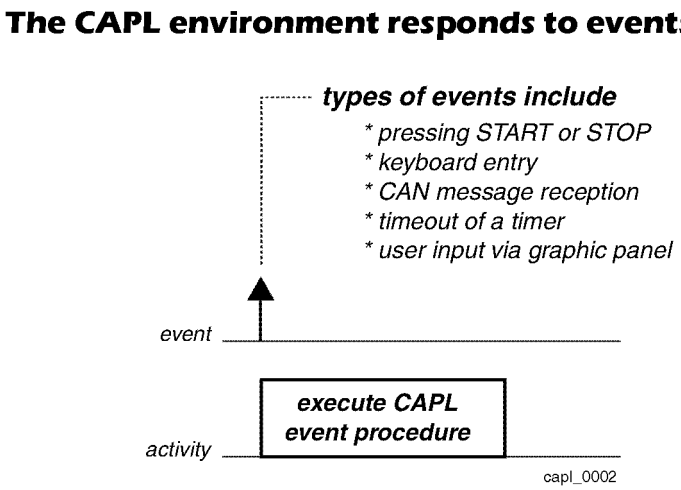
Within this programming environment a user can create, modify, and maintain CAPL programs that can interface with a wide variety of inputs, outputs and other functions. Start-stop events, keyboard entry events, the ability to transmit and receive CAN messages, interaction with the serial port and parallel port, the use of timers, and the ability to interconnect to customer specific DLLs are some of the interface choices available inside the CAPL programming environment.

Most programs are developed using the CAPL Browser, which provides an easy to use "edit-thru-compilation" development process. Even though provided as an integrated component of CANalyzer or CANoe, the CAPL Browser application program can also be used separately.



4.0 CAPL Responds to Events

In general, CAPL programs are simply designed to detect events and then execute the associated event procedure.



For example, you could output the text "Start Test" to the Write window upon clicking on the start button of CANalyzer by using the following program.

```
on start
{
write("Start Test");
}
```

The types of events that can be detected in CAPL include -

- the pressing of the start
- stop button
- user keyboard entry
- CAN message reception
- the timeout of a timer
- user input via a graphic panel (available only in CANoe).

Based on this "event causes procedural activity" technique, CAPL programs are constructed and organized as a collection of event procedures.

The CAPL Browser is conveniently structured into different groupings of event procedures. These so-called event procedural groups or event classes include system, CAN controller, CAN message, timer, keyboard, and CAN error frame as shown in the CAPL event table.

CAPL Events

capl_0006

Event Class	Event Name	Procedure executed
System	*PreStart*	*During initialization of CANalyzer or CANoe*
System	*Start*	*On press of Start button for CANalyzer or CANoe*
System	*StopMeasurement*	*On press of Stop button for CANalyzer or CANoe*
CAN Controller	*BusOff*	*On detection of Bus Off condition in tool hardware*
CAN Controller	*ErrorActive*	*On detection of Error Active condition in tool hardware*
CAN Controller	*ErrorPassive*	*On detection of Error Passive condition in tool hardware*
CAN Controller	*WarningLimit*	*On detection of Warning Limit condition in tool hardware*
CAN-Message	*user defined*	*On reception of designated message*
Timer	*user defined*	*On timeout of designated timer*
Keyboard	*user defined key*	*On keyboard press of designated key*
ErrorFrame	*ErrorFrame*	*On detection of each Error Frame in tool hardware*
Function	*user defined*	*On invocation of the user's designated procedure*
Environment	*user defined*	*On change in designated environmental variable (CANoe only)*

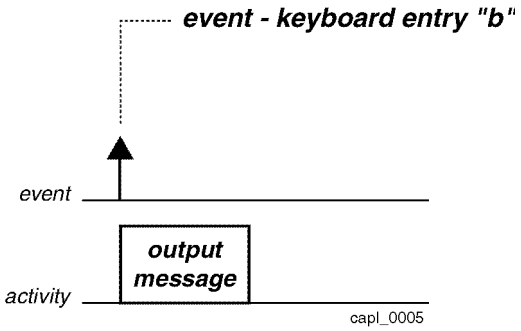
5.0 Example CAPL Programs

Let's take a look at how much CAPL code is necessary to develop the three common information transfer methods (or transfer dialogs) used by most CAN-based distributed embedded systems. Our three examples will include the transmission of an event message, the transmission of a periodic message, and the transmission of a conditionally periodic message.

5.1 Event Message Transmission

When information only needs to be transferred on an event basis, the event message is used. This sample program uses the pressing of the 'b' key on the PC keyboard to initiate a single CAN message transmission.

To send an event message



It is important to note that the use of each message must include a definition within the global variables. In this example we have defined the message msg1 as having a CAN identifier of 555 hex and data length code (dlc) or data size of one byte.

Note - While this example shows defining a message completely within this CAPL program, it is easy to use the CANdb program (another related application program in the Vector CAN tool suite) to link a user-defined, industry defined or corporate specific messaging database.

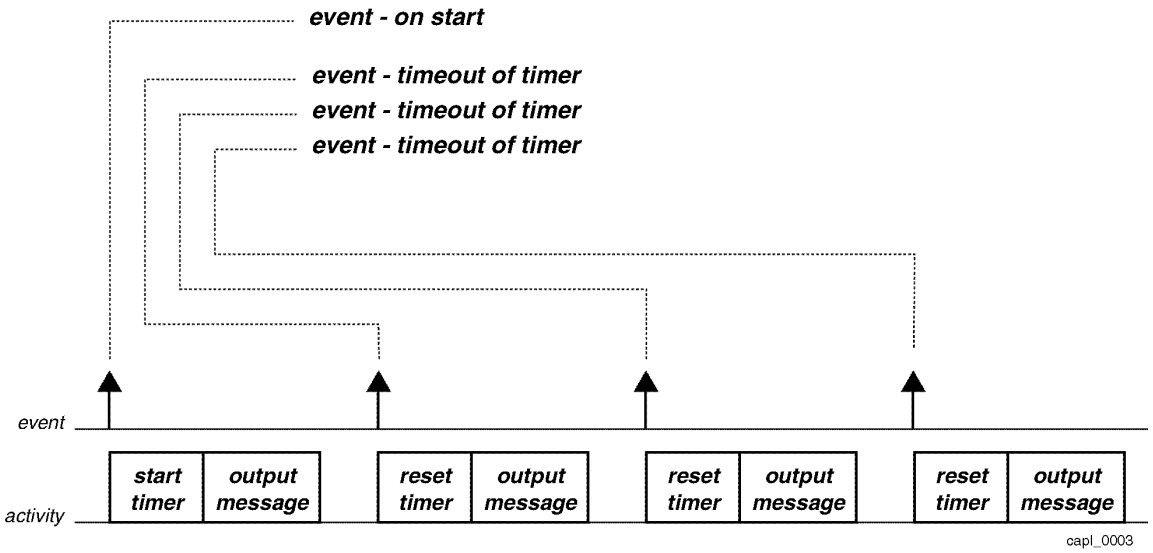
In the "keyboard" event class we insert (via the right mouse button) a new procedure for the 'b' key press. Don't forget to include the single quote marks. In this “on key” procedure, we fix the first byte of the can message data equal to hexadecimal AA and then use the internal CAPL function output to transmit the message. Notice that the message name used in the output function is identical to the name defined in the global variables.

```
 variables
{
message 0x555 msg1 = {dlc=1};
}

on key 'b'
{
msg1.byte(0)=0xAA;
output(msg1);
}
```

6.0 Periodic Message Transmission

When information requires transferring on a repetitive basis, the periodic message is used.

To send a periodic message



To send the periodic message requires the use of a timer and this timer must be defined by name in the global variables area. To get this timer running, we have chosen to initialize the timer in the "on start" event procedure that is executed upon clicking on the tool start button. We need to use one of the intrinsic CAPL functions called setTimer to initialize the time value to 100 milliseconds.

When this timer expires (or times out), the corresponding "on timer" event procedure will be executed. Within this "on timer" event procedure the timer will be reset again to 100 milliseconds. Additionally, we increment the first message data byte before sending the CAN message using the CAPL output function.

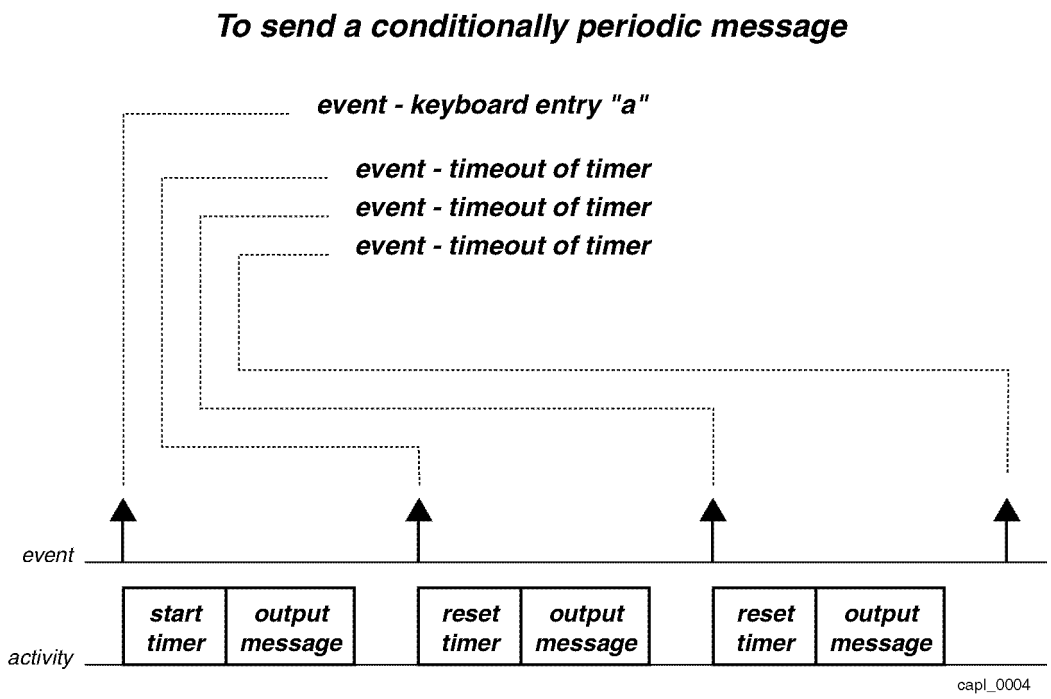
This repetitive process will continue until one clicks on the tool's stop button.

```
variables
{
message 0x555 msg1 = {dlc=1};
mstimer timer1;                          // define timer1
}


on start
{
setTimer(timer1,100);               // initialize timer to 100 msec
}


on timer timer1
{
setTimer(timer1,100);               // reset timer

msg1.byte(0)=msg1.byte(0)+1;        // change the data
output(msg1);                       // output message
}
```

With respect to the CAPL program development, for this periodic message example, CAPL code must be entered into three different locations and a total of 6 lines of code are used.

6.1 Conditionally Periodic Message Transmission

When information requires transferring on a repetitive basis only when a certain set of conditions is true, the conditionally periodic message is used. This CAPL program uses the pressing of the 'a' key on the PC keyboard to toggle between no message transmission and a periodic message transmission.



To send a conditionally periodic message also requires the use of a timer, but this timer does not need to run continuously. We have chosen to activate or deactivate the timer based upon the pressing of the <a> key sequence. The first press of the <a> key will start the periodic message and the second press of the <a> key will stop the periodic message. Each subsequent press of the <a> key will toggle back and forth between these two states.

Notice in the global variables, we have defined a different message and timer name than was used for our periodic message example. This allows you the opportunity to experiment with both message transfer methods inside the same CAPL program if desired.

To properly start the process in the inactive state, we have defined an integer variable called conditionA in the global variables area and have initialize its value to 0 which represents the off condition.

Once the user presses the <a> key, the corresponding on key procedure will execute. This procedure will toggle the condition variable and check to see if the condition is active or true. If the condition is active, then the timer will be started.

When this timer expires, the corresponding "on timer" event procedure will be executed. The timer is reset to continue running and after modifying the data the CAN message is transmitted.

Notice that when the condition has been toggled back to the off condition, it will be the next timer event that actually stops the timer from running.

This conditionally repetitive process will continue until the tool is stopped.

With respect to the CAPL program development, source code must be entered into three different locations and a total of 10 lines of CAPL code are used.

```
variables
{
message 0x400 msgA = {dlc=1};
mstimer timerA;
int conditionA = 0;              // initialize conditionA = off
}


on key 'a'
{
 conditionA =! conditionA;      // toggle conditionA
 if(conditionA == 1)            // if condition is active
   {
   setTimer(timerA,200);        // then start timer
   }
}

on timer timerA
{
 if(conditionA == 1)            // if condition is still true
   {
   setTimer(timerA,200);        // then continue timer
   }
msgA.byte(0)=msgA.byte(0)-1;   // change the data
output(msgA);                  // output message
}
```

7.0 Limitations of CAPL

CAPL is limited more by available PC resources than by its internal architecture and operation.

As an example, high-speed gateways between CAN and other UART-based protocols were used to determine where the boundaries might lie. Laptops were exchanged to see the effects of increased speed, which showed it is quite difficult to deliver consistent qualitative numbers.

Many CANoe models use c.40 concurrently running CAPL programs to simulate an entire car in real time. The model is also capable of interconnecting to real modules or sub-system portions of a real vehicle.

8.0 Conclusions

From design through manufacturing and service, CAPL programs can be quite helpful.

To experiment with CAPL, the demo version of CANalyzer (available from our website) can be used to write, compile, and do real CAPL programming. The resulting code written in the demo may be transferred to a fully licensed tool at a later date.

9.0 Contacts

Vector Informatik GmbH
Ingersheimer Straße 24
70499 Stuttgart
Germany
Tel.: +49 711-80670-0
Fax: +49 711-80670-111
Email: info@vector-informatik.de

Vector CANtech, Inc.
39500 Orchard Hill Pl., Ste 550
Novi, MI 48375
Tel: (248) 449-9290
Fax: (248) 449-9704
Email: info@vector-cantech.com

VecScan AB
Fabriksgatan 7
412 50 Göteborg
Sweden
Tel: +46 (0)31 83 40 80
Fax: +46 (0)31 83 40 99
Email: info@vecscan.com

Vector France SAS
168 Boulevard Camélinat
92240 Malakoff
France
Tel: +33 (0)1 42 31 40 00
Fax: +33 (0)1 42 31 40 09
Email: information@vector-france.fr

Vector Japan Co. Ltd.
Seafort Square Center Bld. 18F
2-3-12, Higashi-shinagawa,
Shinagawa-ku, J-140-0002 Tokyo
Tel.: +81 3 5769 6970
Fax: +81 3 5769 6975
Email: info@vector-japan.co.jp