**Módulo 5 – CAPL Básico**

1. **Introdução**

CAPL é uma sigla para (Communication Access Programming Language), e é uma linguagem orientada a eventos desenvolvida pela empresa Vector Informatik GmbH. Essa linguagem é utilizada no softwares CANoe e CANalyzer, utilizaremos no projeto para o desenvolvimento dos blocos simulados.

1. **O que seria uma linguagem orientada a eventos?**

Como dito anteriormente, CAPL é uma linguagem orientada a eventos, porém o que seria isso? Diferentemente de C, que é uma linguagem estruturada, ou seja, que segue uma sequência. A linguagem CAPL é baseada em “reações” a eventos pré-estabelecidos, ou seja “quando acontecer isso... faça isso”.

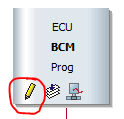
Por esse motivo, todo código o CAPL será desenvolvido em diferentes blocos, que seriam funções chamadas através de gatilhos (*trigger’s*), esses gatilhos por sua vez, podem ser através de Temporizadores, Eventos I/O e Eventos na rede CAN. Isso será abordado posteriormente.

1. **Semelhanças com C**

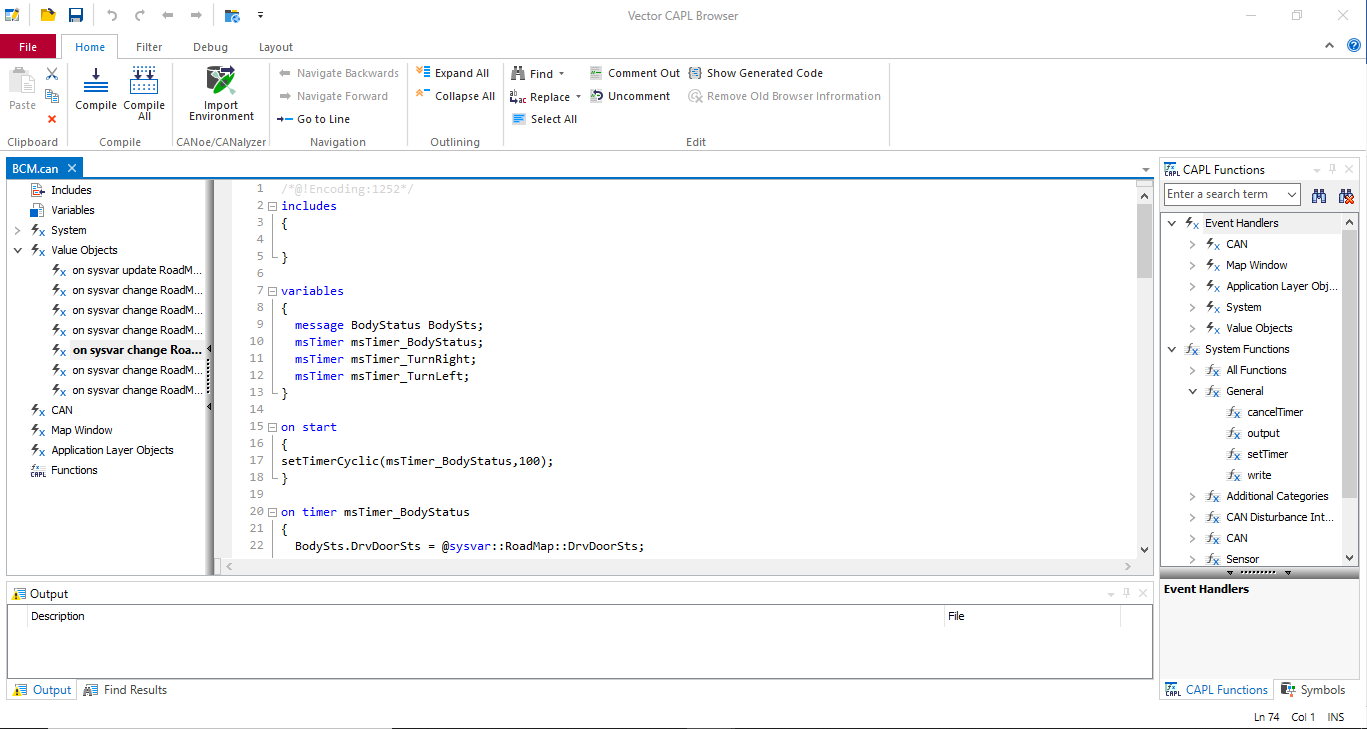
A semântica de CAPL é bastante parecida com C,

1. **CAPL Browser**

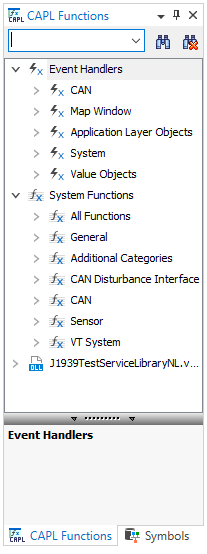
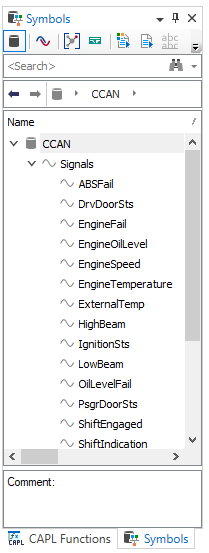
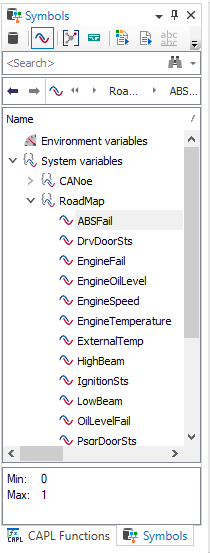
O CANoe possui sua própia IDE, o ‘*CAPL Browser’*, esta é uma ferramenta muito útil, pois ajuda muito durante o desenvolvimento do código, enquanto o código está sendo escrito a própia IDE da sugestões baseadas no DBC que está sendo utilizado na simulação. Por esse motivo, é sempre importante abrir o *CAPL Browser* através do CANoe, para que o mesmo associe o DBC presente na simulação. Para isso, basta entrar na parte de *Simulation* do CANoe e clicar no “lápis” presente no bloco da simulação.



Após clicar no lápis, o *CAPL Browser* abrirá com o DBC já associado.



O *CAPL Browser* também facilita o desenvolvimento do código através da barra presente no canto esquerdo, com ela é possível selecionar os ‘**Eventos**’, ‘**Sinais** e ‘**Variáveis de Sistema**’, e com o mouse, apenas arrastar para a área de código.

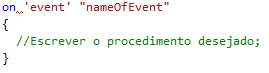
  

1. **Criando um evento**

Como dito anteriormente, o CAPL é uma linguagem orientada a eventos. Nessa seção abordaremos os principais eventos utilizados na construção de um simulação, citados abaixo:

* **on message**
* **on timer**
* **on sysvar**
* **on key**
* **on preStart**
* **on start**
* **on stopMeasurement**
* **on errorFrame**

A estrutura básica de um evento é a seguinte:



É importante não se esquecer de fechar as chaves, e também de finalizar cada linha de código com ponto e vírgula, para evitar erros de compilação.

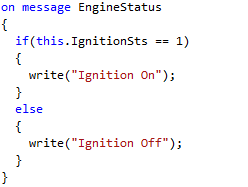
Abordaremos agora cada um dos eventos citados, quando e como eles são utilizados no código, inicialmente iremos focar apenas nos eventos, e posteriormente abordaremos algumas funções nativas de CAPL.

**on message** *‘MessageName’***:**

Evento utilizado quando é necessário executar uma ação ao receber uma mensagem específica na rede.

**Exemplo:** Ao receber a mensagem EngineSts, verificar se a ignição está ou não ligada.

**Code:**

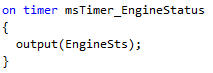


**on timer** *‘TimerName’***:**

Evento utilizado quando é necessário executar uma ação após um tempo determinado, seja ou não-cíclico.

**Exemplo:** Enviar uma mensagem cíclica na rede a cada 100 ms.

**Code:**

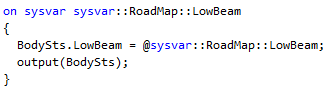


**on sysvar** *‘SysvarName’***:**

Evento utilizado quando é necessário executar uma ação após a mudança de valor em uma variável de sistema.

**Exemplo:** Ao ativar um botão no painel, acionar o faról baixo.

**Code:**

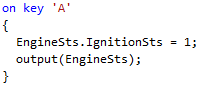


**on key** *‘Key’***:**

Evento utilizado quando é necessário executar uma ação após acionar algum botão do teclado.

**Exemplo:** Ligar a ignição após apertar a tecla ‘A’

**Code:**



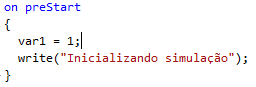
**Importante sempre lembrar que CAPL é uma linguagem *‘Case Sensitive’*, ou seja, letras minúsculas e maiúsculas são diferentes.**

**on preStart:**

Evento utilizado quando é necessário executar uma ação durante a inicialização da simulação.

**Exemplo:** Inicializar uma variável e escrever uma frase na tela “Write”.

**Code:**

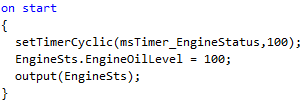


**on start:**

Evento utilizado quando é necessário executar uma ação assim que a simulação é iniciada.

**Exemplo:** Inicializar timers, mudar valores de sinais e enviar mensagens na rede.

**Code:**

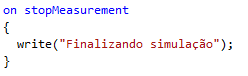


**on stopMeasurement:**

Evento utilizado quando é necessário executar uma ação quando a simulação é pausada.

**Exemplo:** Escrever na tela “Write” ou escrever em um arquivo log.

**Code:**

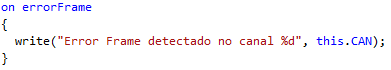


**on errorFrame:**

Evento utilizado quando é necessário executar uma ação ocorre um *‘Error Frame’* no barramento.

**Exemplo:** Escrever na tela “Write” o barramento que ocorreu um erro.

**Code:**



1. **Declarando e inicializando Temporizadores**

Os temporizadores, mais conhecidos como *‘timers’*, são uma das ferramentas mais úteis quando se trata de desenvolvimento de simulações, com eles é possível o envio de mensagens cíclicas no barramento CAN.

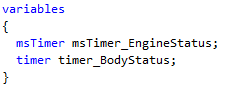
A utilização de um temporizador segue basicamente os 3 passos a seguir:

* Declaração do temporizador
* Inicialização do temporizador
* Evento a ser executado após timer expirado

**Declaração:**

Existem dois tipos de temporizadores em CAPL, o ‘*msTimer*’ e o ‘*Timer*’, a única diferença entre eles é que um trabalha em tempos em milisegundos e outro em segundos.

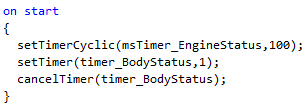
Segue o código abaixo com a declaração dos timers:



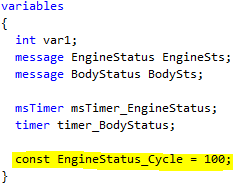
**Inicialização:**

Após declarado o temporizador, ele pode ser inicializado para funcionar apenas uma vez ou ciclicamente, também pode ser reiniciado ou cancelado a qualquer momento. Para isso usamos as seguintes funções:

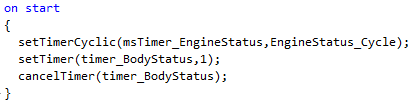
* setTimer(*timer,tempo*)
* setTimerCyclic(*timer, tempo*)
* cancelTimer(*timer*)



Quando estamos lidando com mensagens cíclicas, é deboa prática declararmos o tempo cíclico da mensagem como constantes na seção superior do código.

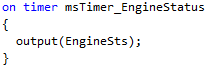


Reformulando o caso anterior, ficariamos com o seguinte código:

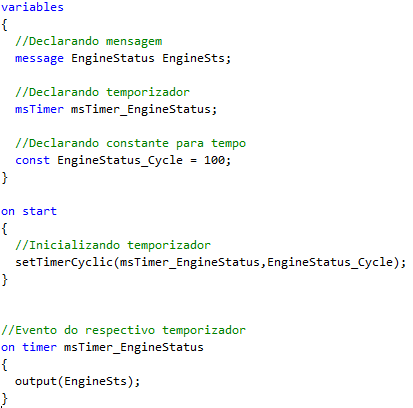


**Evento:**

Utilizando a função **“on timer”**, a simulação executará os comandos dentro do evento do respectivo timer



Todos as partes para a utilização de um temporizador foram mostradas e explicadas separadamente, segue um exemplo de código de uma implementação completa de um temporizador responsável pelo envio cíclico da mensagem “*EngineSts*”:



1. **Funções Utéis CAPL**

Como dito anteriormente, existem diversas funções internas em CAPL, abordarei a

* write
* output
* startLogging
* stopLogging