VECTOR >

はじめての CAPL

ビギナー向け資料「はじめてシリーズ」

vector.com

目次

発行元：ベクター・ジャパン株式会社

はじめに

本稿は、CANoe/CANalyzer の CAPL 機能について初歩的な内容をまとめたものです。「プログラミングははじめて」、「C 言語がよく分からない」という方でも、本稿を読めば CAPL が使えるようになることを目指して作成いたしました。CAPL ノードの作成方法から基本的なメッセージ※1送信方法、受信メッセージに対する反応の記述、データ値の変更など、CAPL の基礎をご紹介していきます。ここでは CAN 通信を前提として説明しておりますが、オプションにより他のプロトコル（LIN・Ethernet・FlexRay など）を使用の場合にも一部ご参考いただけます。

なお、ベクター・ジャパン Web サイトの「CAPL の使用」では、効率的でパフォーマンスに優れたプログラムを CAPL で実装する際のヒントやコツを記載しております。詳細は以下リンク先をご覧ください。（ベクターの Web サイトへ移動します）

https://www.vector.com/jp/ja/know-how/capl/

また、CAPL の実践的な知識についてお知りになりたい方には、ベクター・ジャパンにて CAPL 関連のトレーニングも開催しておりますので、そちらも是非ご利用ください。以下から、内容や日程を確認できます。

https://academy.vector.com/jp/ja/

【重要な注意事項】

・本稿は CANoe pro および CANalyzer pro の各機能を基準に説明しています。

・CANoe run、pex および CANalyzer fun、exp については、機能制限がございます。特に、CAPL プログラムの新規作成／編集には CANoe pro または CANalyzer pro が必要です。機能制限の詳細につきましては、CANoe/CANalyzer のオンラインヘルプ、もしくは以下リンク先から「機能マトリックス」をご覧ください。

https://cdn.vector.com/cms/content/products/canoe/canoe/docs/Fact%20Sheets/CANoe_CANalyzer_FeatureMatrix_JP.pdf

※1 本稿では、CAPL 関数名との不一致による誤解を防ぐため、データ伝送の構成単位である“フレーム”を“(CAN)メッセージ”として表現します。

1. CAPL とは？

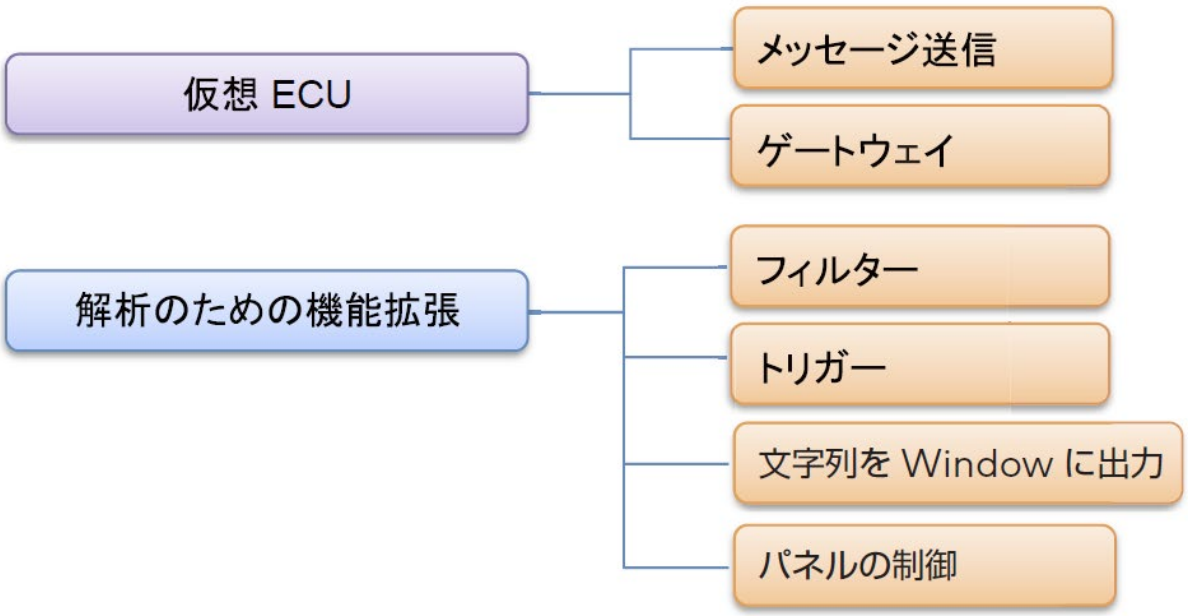
CAPL（キャプル）は Communication Access Programming Language の略称であり、CANoe/CANalyzer 専用のプログラミング言語です。

2. CAPL の特徴

- C 言語に似ている
- 「何かが起きた時に何かを行う」イベントドリブンのプログラム実行形式で動作する

3. CAPL の役割

CAPL は、バス上に存在する 1 つの仮想ノードとしてメッセージ送受信を行うことができます。CANoe では、この仮想ノードを複数作成することができるので、仮想ネットワークのシミュレーションを行うことが可能です（CANalyzer は 1 つのみ）。また、異なるバス間のゲートウェイとして使用したり、フィルターやトリガーといった解析のための機能拡張に用いたり、さまざまな役割をすることができます。その他、CANoe 専用の機能として、自動テストのモジュールを実装することができます。



4. CAPL ノードの設置

CAPL を配置するためのノード(ブロック)を設置します。CAPL ノードの設置位置は任意ではなく、プログラムの機能に合わせ、考えて配置されなければなりません。

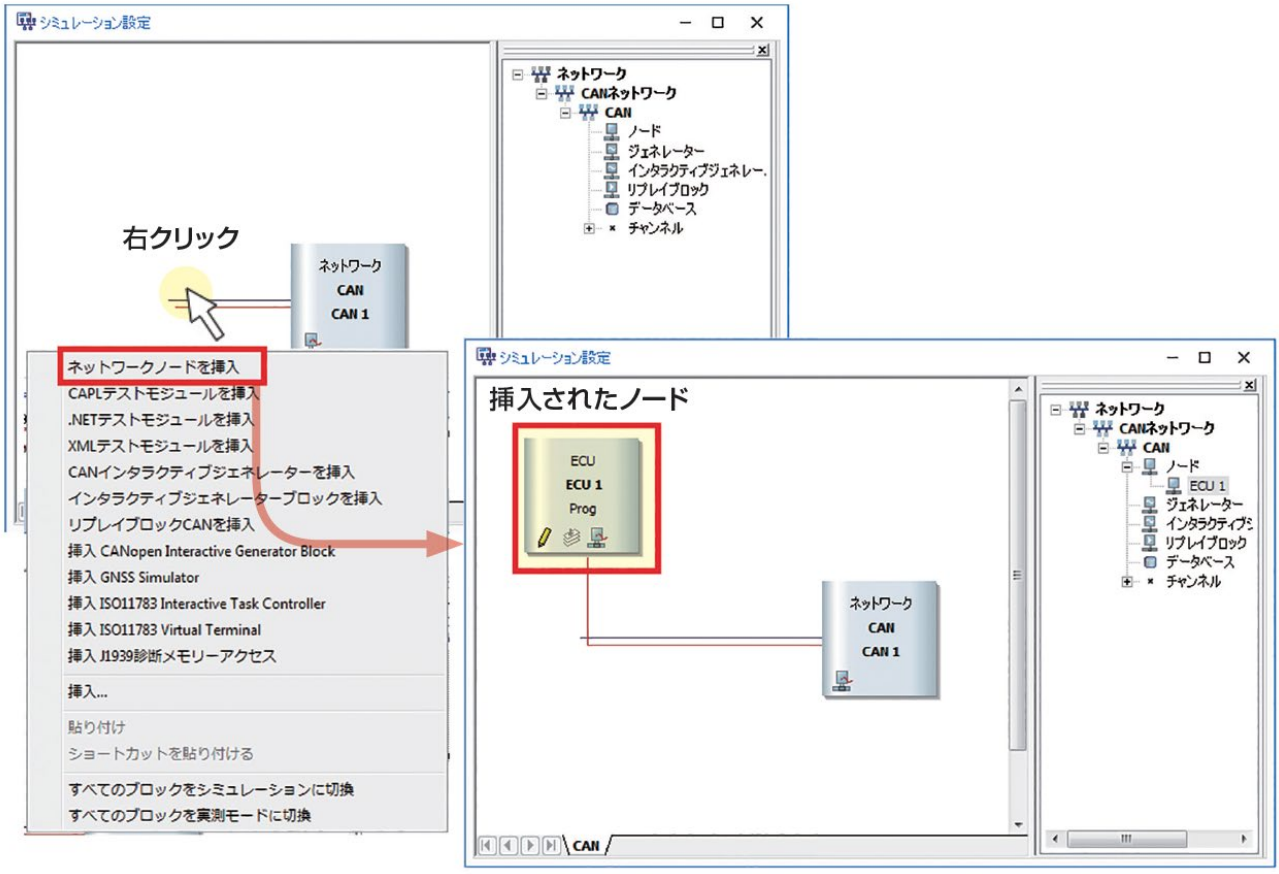
以下に、実バスにメッセージ送信を行うためのノード設置を説明します。

4.1 CAPL ノード挿入

4.1.1 CANoe の場合

CANoe では、シミュレーション設定 Window に設置します。

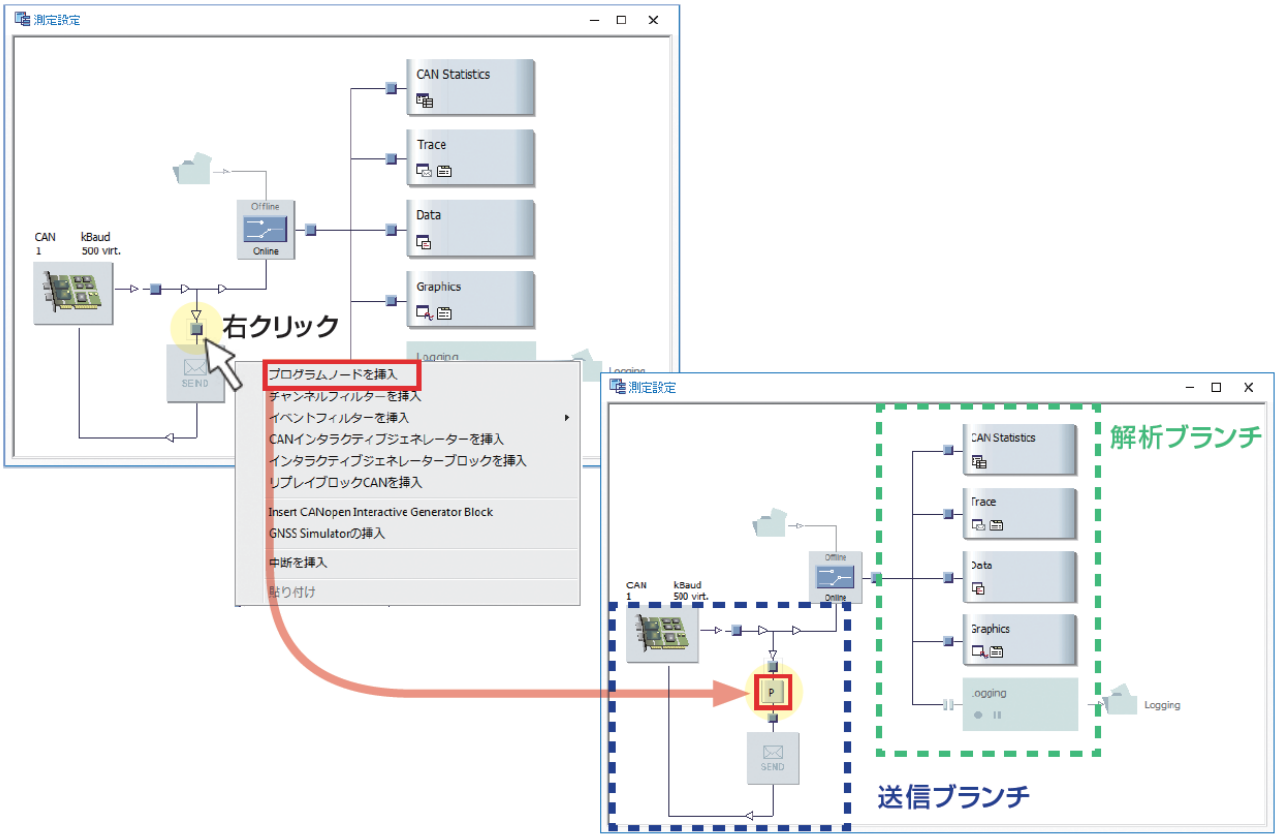
1. CANoe のリボンより[シミュレーション]-[シミュレーション設定]を選択します。
2. シミュレーション設定 Window の赤黒 2 重横線の上で右クリックして、コンテキストメニューより[ネットワークノードを挿入]を選択します。



4.1.2 CANalyzer の場合

CANalyzer では、測定設定 Window に設置します。

1. CANalyzer のリボンより[解析と刺激]-[測定設定]を選択します。
2. 測定設定 Window の送信ブランチのホットスポット( )の上で右クリックして、コンテキストメニューより[CAPL ノードを挿入]を選択します。



3. メッセージを実バスに送信するための CAPL プログラムは、測定設定 Window の「送信ブランチ」に挿入します。

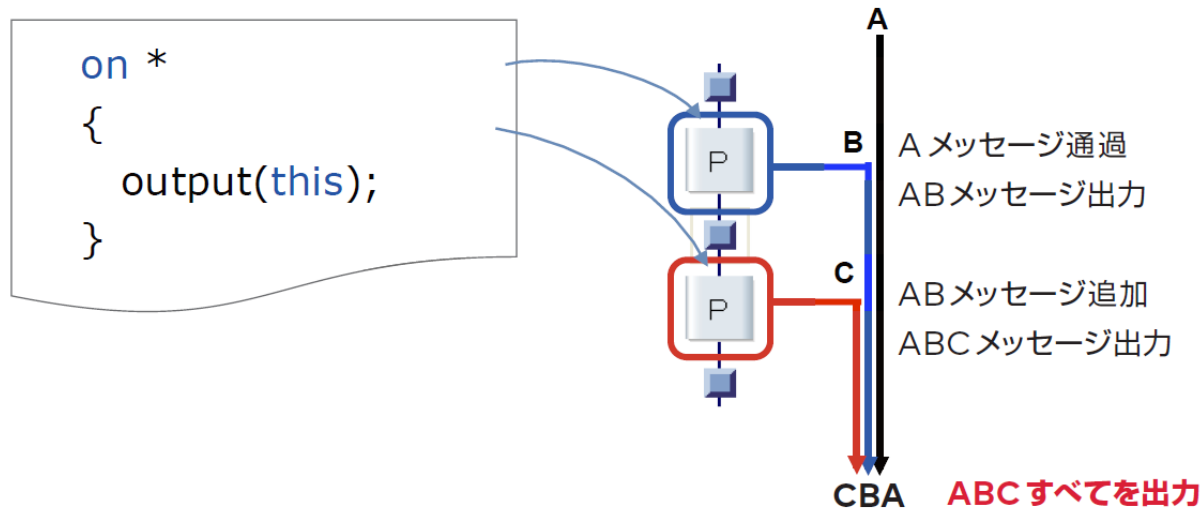
4.2 CAPL 配置に関する注意

CAPL ブロックを直列に配置した場合、下流にある CAPL ノードはフィルターのような動作をします。そのため、上流のメッセージは下流の CAPL ブロックに遮られ、メッセージを通過させません。CANalyzer の場合、測定設定 Window の送信ブランチ内に（下図のように）直列配置すると、上から流れてきたメッセージはすぐ下の CAPL ブロックにより遮断されます。

メッセージの遮断はすべてのメッセージ通過を明示的に出力するプログラムを実装することにより、回避することができます。

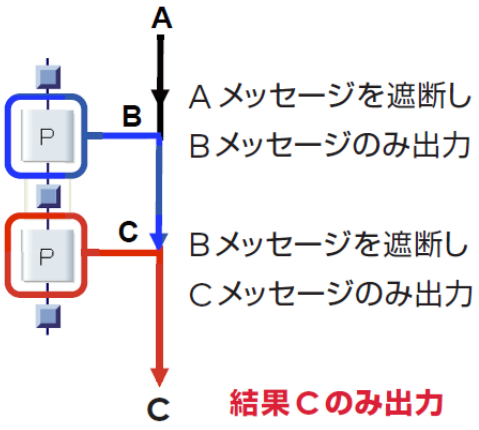
【出力プログラム】 重要!!

```
on *
{
  output(this);
}
```

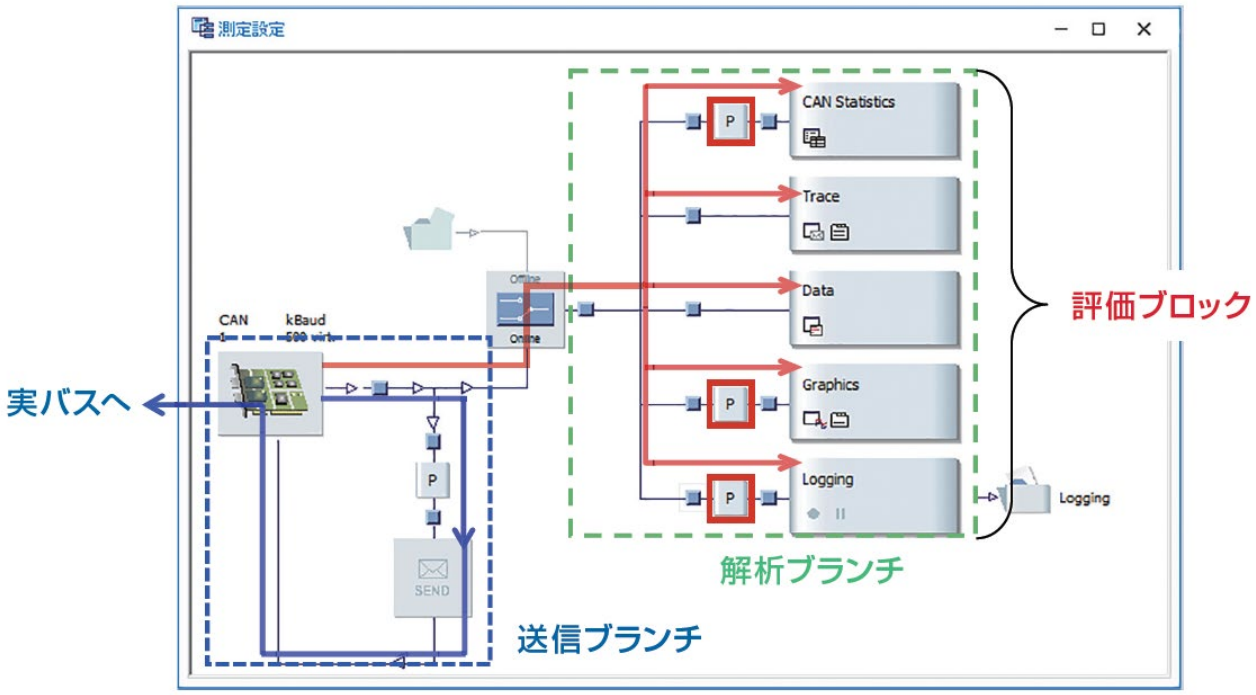


※注意

上記プログラムを実装しないと
C のメッセージしか出力されません!!



メッセージを実バスに送信する必要のないプログラム（データの解析にのみ使用する CAPL プログラム）は、測定設定 Window の評価ブロック前に設置することを推奨します。

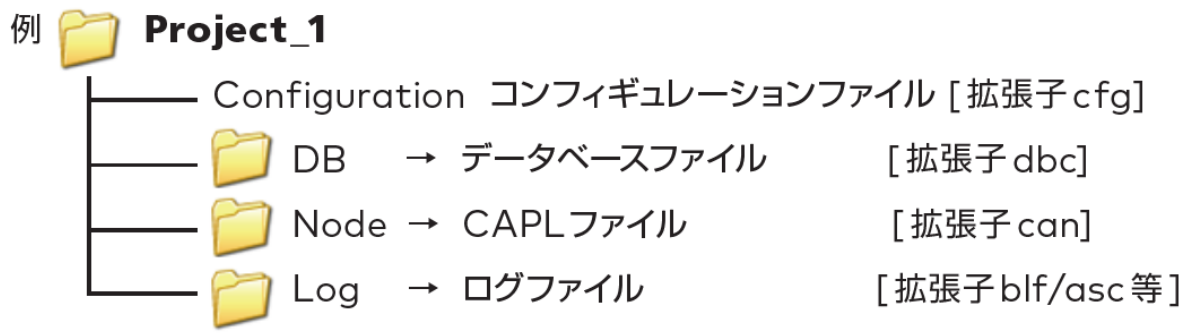


上図は CANalyzer の測定設定 Window ですが、各評価ブロックへの流れは CANoe も同様です。CANalyzer の送信ブランチは、CANoe ではシミュレーション設定 Window になります。

※ CANoe pex の場合、機能制限により一部の評価ブロックが無効になるため、設置した CAPL の実装が無効になる場合があります。

4.3 ファイル管理に関する注意（参考）

CANoe/CANalyzer のファイル管理は、以下の基本構成でフォルダ管理すると便利です。



[Project_1]と名前付けしたフォルダを大きなくくりとし、その中に[DB]と[Node]という 2 つのフォルダをあらかじめ作成します。そして、このコンフィギュレーションで使用するデータベースを[DB]にあらかじめ保存します。作成した CAPL ファイル（拡張子 can）の保存先は、このあらかじめ作成しておいた[Project_1]-[Node]を指定し、名前を付けて保存します。

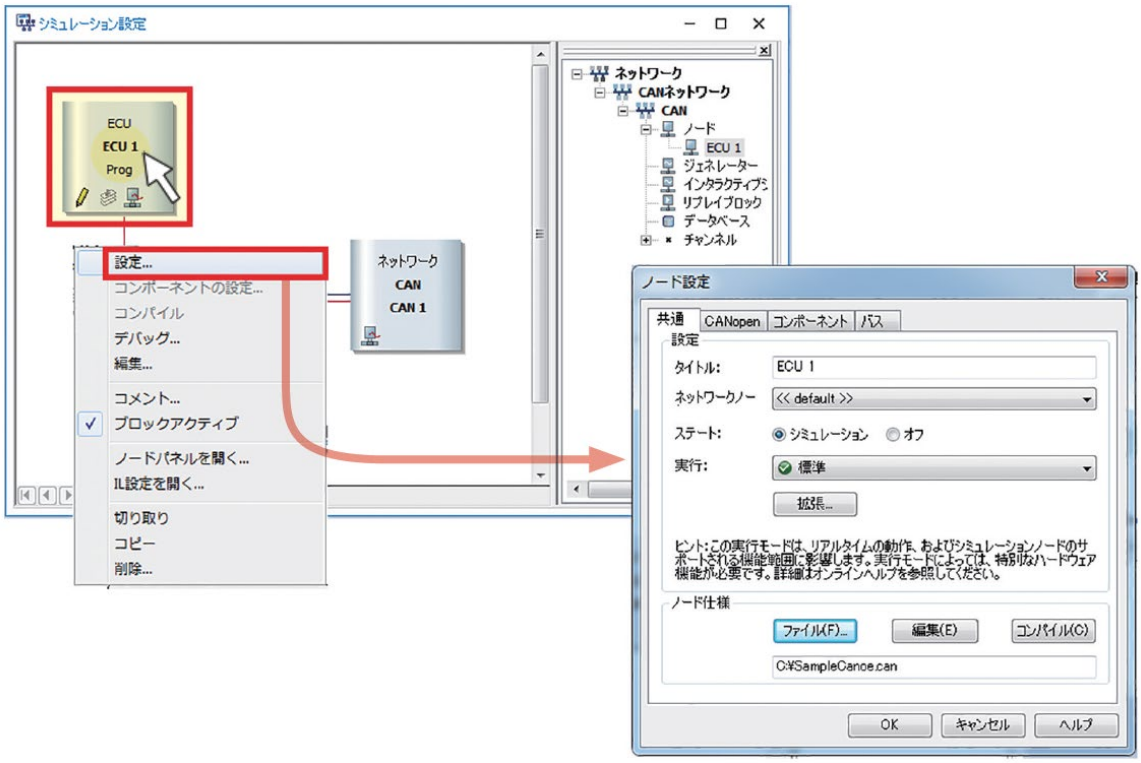
こうしておくことで、使用する PC が変更になった場合も、[Project_1]をコピーして一括で移管することが可能となり、一部ファイル不足やコンパイル時のエラーを回避することができます。

5. CAPL ファイルの作成

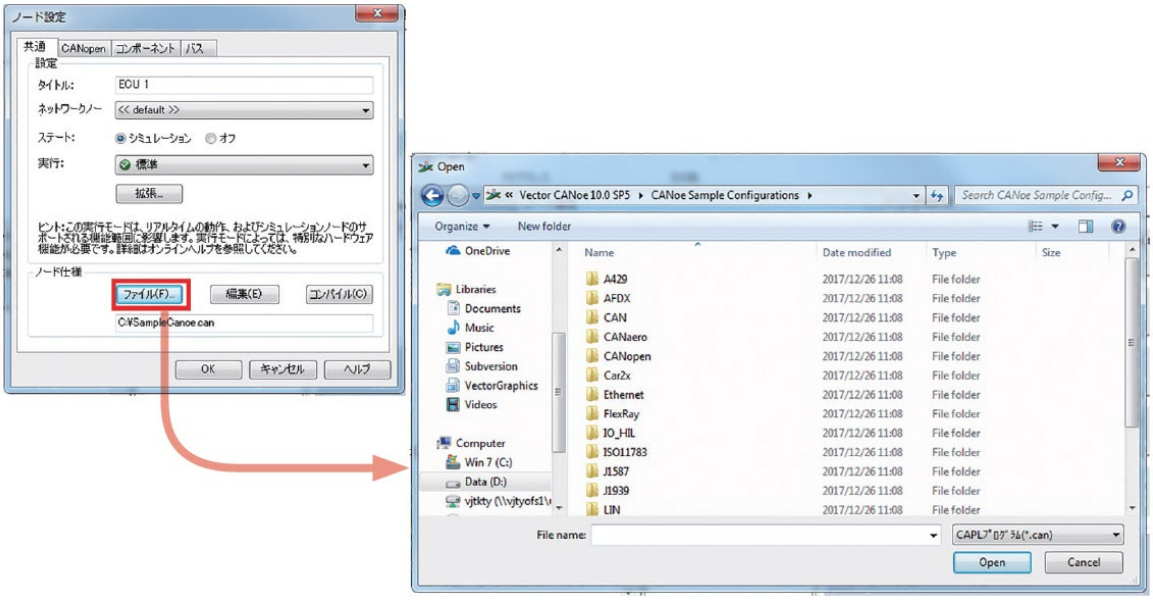
CAPL ファイルの保存先と名前を決定します。

5.1 CANoe の場合

1. ノードを右クリックして[設定]を選択し、[ノード設定]を開きます。

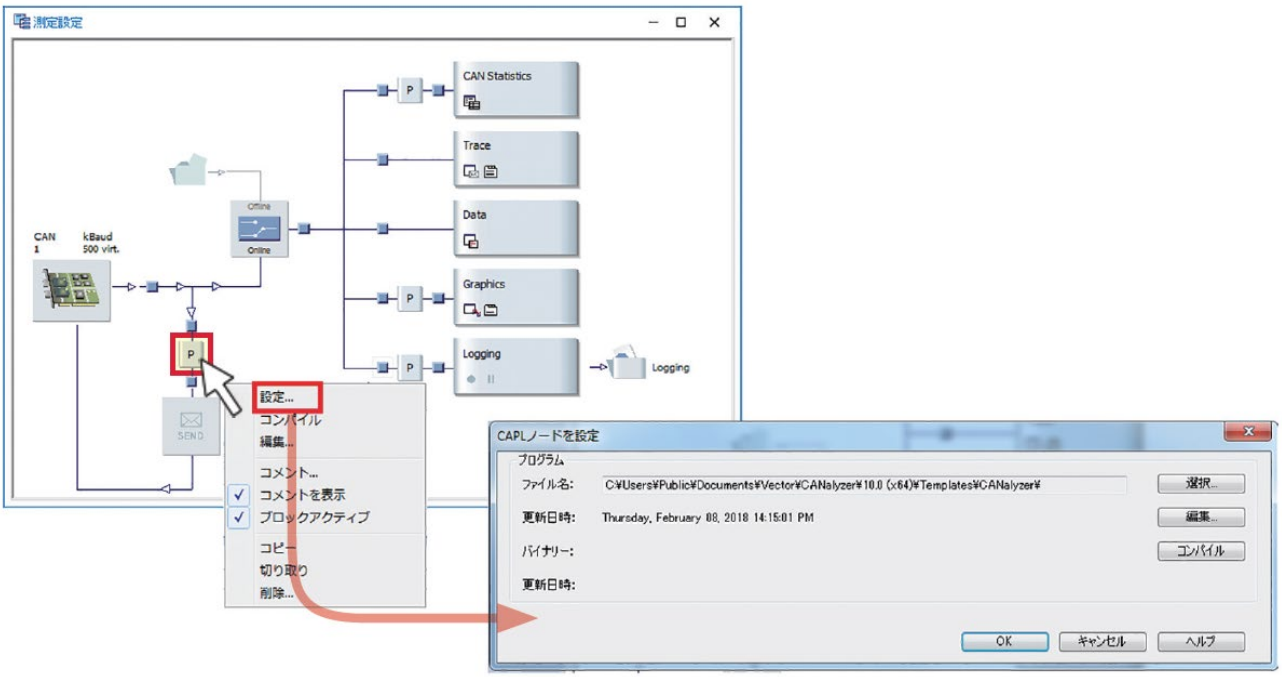


2. [ノード設定] の[ファイル]をクリックして [ファイルを開く] 画面を表示し、CAPL ファイルの保存先と名前を決定します。

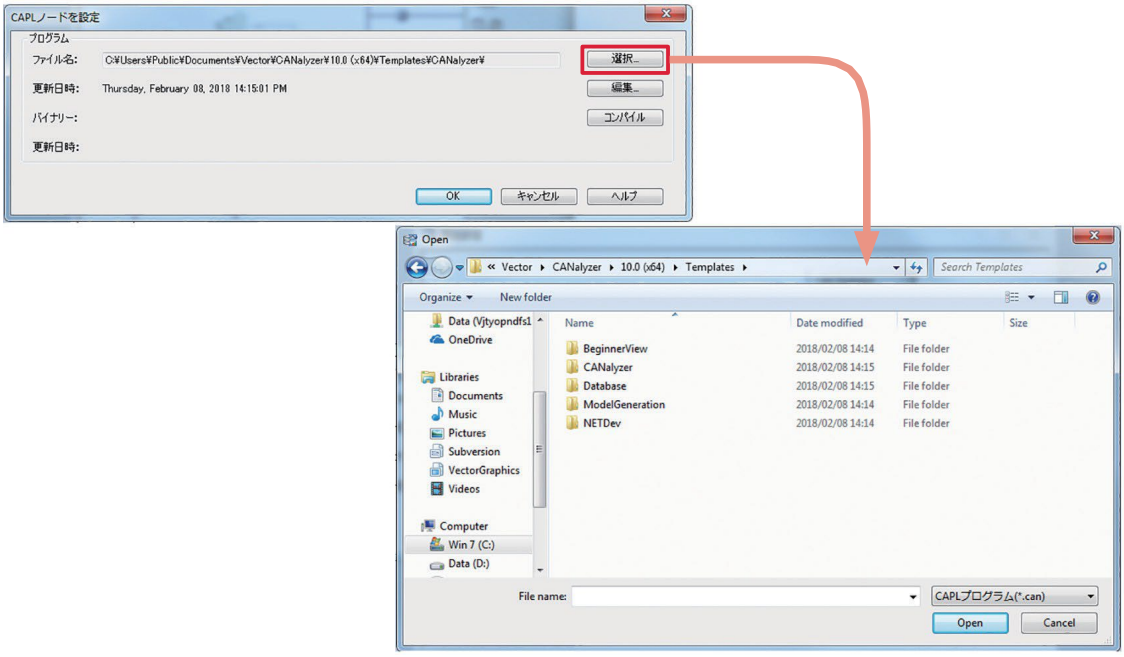


5.2 CANalyzer の場合

1. ノードを右クリックして[設定]を選択し、[CAPL ノードを設定]を開きます。



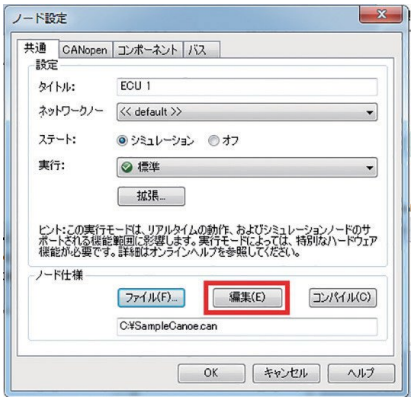
2. [CAPL ノードを設定] の[選択] をクリックして [ファイルを開く] 画面を表示し、CAPL ファイルの保存先と名前を決定します。



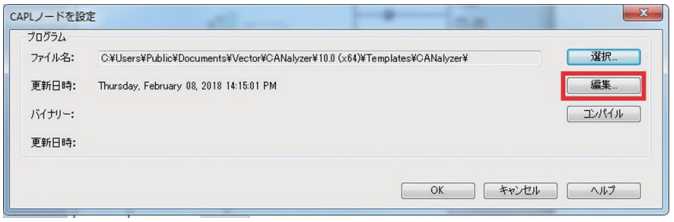
6. CAPL ブラウザーの起動

[編集]をクリックすることにより、CAPL ブラウザーが起動します。

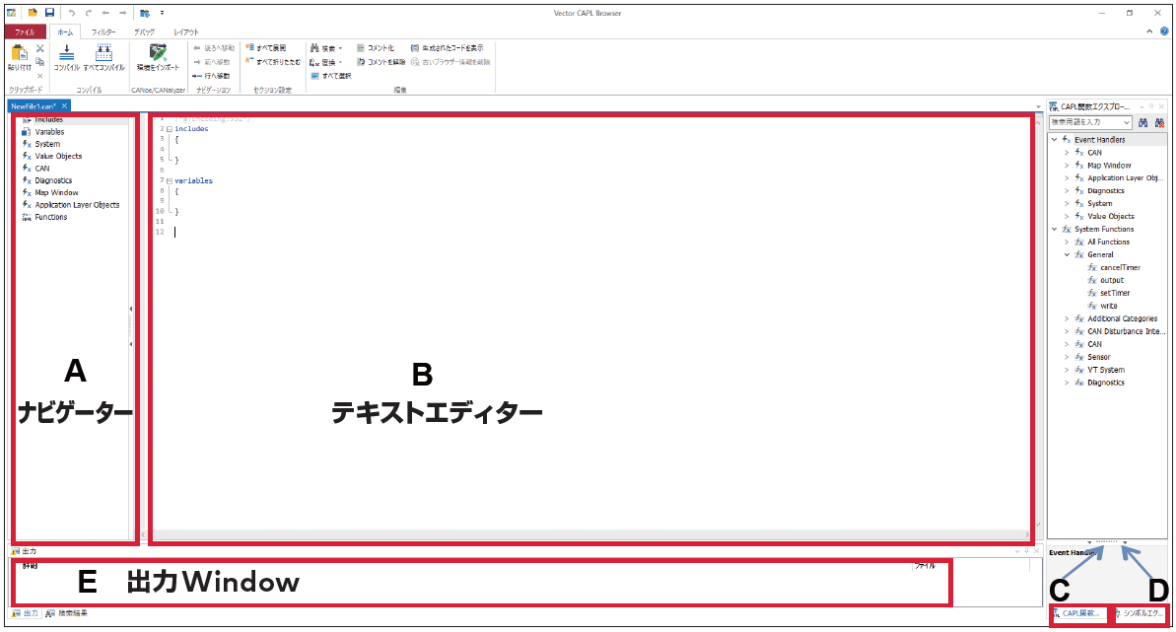
CANoe の場合



CANalyzer の場合



【CAPL ブラウザーの画面】（基本画面は CANoe/CANalyzer 共通です）



A	ナビゲーター	CAPLプログラムの構造をツリーで表示します
B	テキストエディター	CAPLプログラムのソースコードを記述します
C	CAPL関数エクスプローラー※	CAPLのイベントハンドラーとCAPL関数の一覧を表示します
D	シンボルエクスプローラー※	シグナルをテキストエディターにドラッグ＆ドロップで挿入可能
E	出力Window	コンパイルの結果が表示されます

※ CAPL 関数エクスプローラー（C）とシンボルエクスプローラー（D）はタブ切り替えで表示

7. CAPL の記述

7.1 CAPL の用語

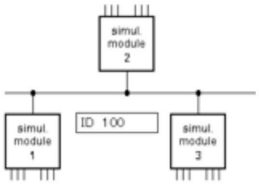
「何かが起きた時に何かを行う」イベントドリブンのプログラム実行形式で動作する CAPL には、イベントハンドラーというものがあります。またプログラミングでは、関数・変数という言葉をよく耳にしますが、CAPL にも関数・変数というものがあります。

以下に、これらの用語(「イベントハンドラー」、「関数」、「変数」)について説明していきます。

7.1.1 イベントハンドラーとは

CAPL でいう「イベント」とは、ユーザーによるキーボード操作やメッセージの受信などをいい、「イベントハンドラー」とは、発生したイベントを検知して何らかの処理を実行させる機能のことをいいます。CAPL の特徴である「何かが起きた時に何かを行う」の、『何かが起きた時』に呼ばれるのがイベントハンドラーです。CAPL は、イベントハンドラーによって発生したイベントに対応して処理を行います。

バスイベント



キーボードイベント

タイマーイベント

これらの「イベント」が発生したら、イベントハンドラーが呼ばれます。それぞれのイベントハンドラーは個々に独立しており、変数と関数によって連結されプログラム化されます。

7.1.2 関数とは

関数(英:function)とは、データを受け取り、決められたとおりの処理を実行するさまざまな命令です。プログラムは、関数を組み合わせて記述します。CAPL には専用の関数があります。

7.1.3 変数とは

変数(英:Variable)とは、プログラムで扱われるデータを一定期間保管して必要なときに利用できるようにするための「名前の付いた箱」のようなもので、この箱から、利用するときに値を持ってくることができるものです。CAPL では C 言語等に用いられる一般的な変数と、CAPL 専用の変数があります。

7.2 CAPL のイベントハンドラー

CAPL は、イベントハンドラーによって発生したイベントに対応して処理を行います。

イベントハンドラーには次のようなものがあり、それぞれのイベントは下記のようになっています。

青字については、本稿の第 7 章～8 章で実際の使用例を説明します。なお、下記のリストの CAN イベントハンドラーは CANoe/CANalyzer の CAN プロトコルで使用します。プロトコルオプション（Ethernet/LIN/MOST/FlexRay/J1939 等）により追加イベントがあります。

イベントハンドラー	イベント
CAN	
on busOff	**CANコントローラーがバスオフへ変化**
on errorActive	CANコントローラーがエラーアクティブ状態へ変化
on errorFrame	エラーフレームの受信
on errorPassive	CANコントローラーがエラーパッシブ状態へ変化
on message <CANID>	**指定したCANメッセージの受信**
on warningLimit	CANコントローラーがワーニングリミットへ到達
on *	メッセージ／フレームの受信（指定なし）
System	
on key <Key>	**キーボードのキーを押す**
on preStart	測定初期化（測定開始前）
on preStop	測定初期化（測定停止前）
on start	**測定開始**
on stopMeasurement	測定停止
on timer	**設定したタイマー時間経過**
Value Objects	
on signal <signal>	シグナルの変化
on signal_update <signal>	シグナルの更新
on sysvar <sysvar>	システム変数(※1) の変化
on sysvar_update <sysvar>	システム変数の更新
AUTOSAR PDU(※2)	
on PDU <PDU name>	**指定したPDUの受信**

※1：システム変数とは、コンフィギュレーション（.cfg）に定義し使用できる CANoe/CANalyzer 専用の変数です。

※2：AUTOSAR により、CAN メッセージと CAN シグナルの間に PDU の概念が追加されました。CAPL ではこの PDU についても送受信可能です。なお、この PDU に関連した CAPL の実装を行うためには AUTOSAR データベースファイル(*.arxml)が必要です。

7.3 CAPL の関数

CAPL の特徴である「何かが起きた時に何かを行う」の『何かを行う』の部分で使われるのが CAPL 関数です。数多くの CAPL 関数がありますが、以下によく使われる基本的なものを一部抜粋しました。これらについては、本稿で後ほど説明します。

CAPL関数	特徴（どのような時に使われるか）
write	出力Windowへテキスト出力 （C言語関数printf ()と同等）
setTimer	タイマーを設定（本稿第7.5.3章を参照）
cancelTimer	タイマーを取り消す（本稿第8章を参照）
output	メッセージ変数を出力（本稿第7.5章で使用しています）
stop	測定を終了（本稿第8章を参照）
triggerPDU	PDU変数を送信（本稿第7.5.4章で使用しています）

7.3.1 関数 write()を使ってみましょう ~演習~

CAPL は、C 言語と同じように、{ }で囲まれたステートメント（命令文）と呼ばれる構成ブロック（1 行以上のセミコロンが後ろに付くコード）を使用してプログラムします。

CAPL は基本的にイベントハンドラーの命令文で構成されており、指定したイベントが発生することにより実行されます。

1. CAPL ブラウザーでプログラムを記述
2. コンパイルする
3. 保存する
4. 実行する

以上の CAPL 記述の基本的な流れを、体感いただければと思います。

CANoe/CANalyzer の測定開始時に、出力 Window で名前表示させてみましょう。

イベントハンドラーは on start　　CAPL 関数は write を使用します

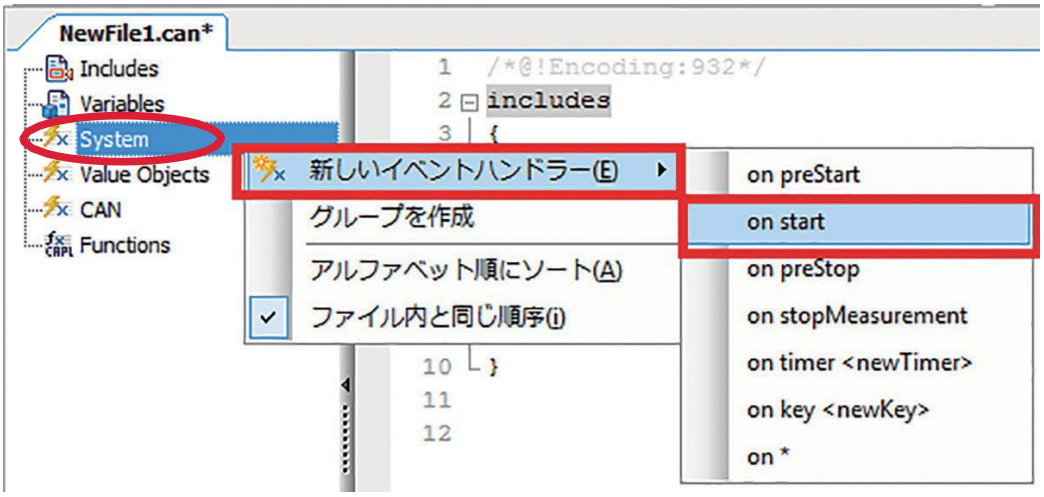
以下を参考に、CAPL 記述の前に CAPL ブラウザーの起動までを終了しておきます。

本稿第 4 章 CAPL ノード挿入

本稿第 5 章 CAPL ファイルの作成

本稿第 6 章 CAPL ブラウザーの起動

① CAPL 起動で説明した CAPL ブラウザー画面左のナビゲーター（区分 A）より、[System] を右クリックし [新しいイベントハンドラー] - [on start] を選択し、テキストエディター（区分 B）に on start を追加します。



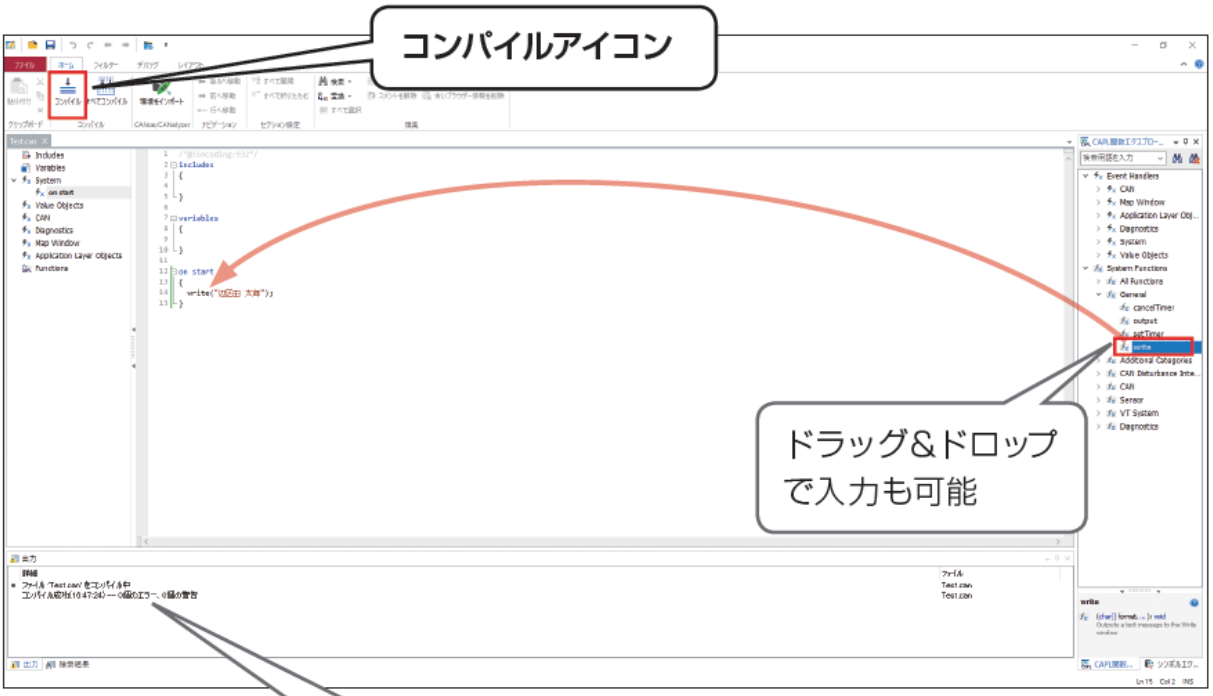
② CAPL ブラウザーの画面中央、テキストエディター（B 部分）に追加された on start の{ } 内に write 関数を用いて以下の例のように記述します。

```
on start
{

        write("辺区田 太郎");

}
```

出力 Window に表示させる文字を ( ) 中の""内に記述します。

最後に; を付けるのを忘れずに!

③　ツールバーのアイコンまたはキーボードの F9 を押下して、コンパイルします。



CAPLブラウザー下方の出力Windowには、コンパイル結果が表示されます
コンパイル成功「コンパイル成功・・」と表示
コンパイル失敗「コンパイル失敗・・・」と表示
コンパイル失敗の場合はプログラムの内容を確認してください。以下に失敗例と対策を記載します。
例）日本語入力で改行　→　改行文字を削除します
；/ “”などが抜けている　→　手順②に示したとおり、プログラムを適切に記述します

④　CANoe/CANalyzer で測定開始し、出力 Window に表示されたことを確認しましょう。

7.4 CAPL で使用される変数

CAPL で使用できる変数には、3 つの種類があります。

1. 一般的な変数（C 言語などで使用されるもの）
2. CAPL 固有の変数
3. システム変数

それぞれの特徴は、以下のとおりです。

変数の種類	定義する場所	特 徴
変数（一般的なもの）	CAPLブラウザー	「整数」「浮動小数点」「文字列」のデータ型がある
CAPLの変数	CAPLブラウザー	「メッセージ」と「タイマー」のデータ型がある
システム変数 (※1)	コンフィギュレーション	CANoe/CANalyzerのcfgファイルに保存される

※1: システム変数は、CANalyzer fun では扱えません。
システム変数にアクセスする際は、「@Namespace::Variable」のように「@」を使用します。

CAPL ブラウザーに定義する変数は、定義した場所により「グローバル変数」または「ローカル変数」として使用されます。グローバル変数とローカル変数の特徴は、以下のとおりです。

変数の種類	定義する場所	使用できる場所	特 徴
グローバル変数	Variables{ } セクション内	CAPLプログラム全体	CANoe/CANalyzerの測定が開始され、CAPLプログラムが実行された時に初期化(※2)
ローカル変数	各イベントハンドラー内	定義したイベントハンドラー内でのみ (※1)	定義されているイベントハンドラーのプログラムが最初に実行された時に初期化(※2)

※1： ローカル変数は、他のイベントハンドラーでは使用できません。定義されたハンドラー以外はその変数を見つけることができませんので、使用するとコンパイルエラーとなります。

※2： CAPL の変数は、初期値を指定しない限り'0'に初期化されます。そして、測定中は新しい値が設定されるまで変数は割り当てられた値を保持します。値を保持したくない（イベントハンドラーが呼び出されるごとに初期値を設定したい）場合は、変数宣言の前にキーワード"stack"を使用します。

7.4.1 CAN における変数のデータ型

データ型	用 途	名 前	ビット(バイト)	補 足
CAPLで使用できる一般的な変数				
整数	符号付	Int	16 bit (2 byte)	-32765...+32767 C/C++:short
		long	32 bit (4 byte)	-2147483648...+2147483647 C/C++:long
		Int64	64 bit (8 byte)	-9223372036854775808 ～ 9223372036854775807 C/C++:long long
	符号なし	byte	8 bit (1 byte)	0...255 C/C++:unsigned char
		word	16 bit (2 byte)	0...65535 C/C++:unsigned short
		dword	32 bit (4 byte)	0...4294967295 C/C++
		qword	64 bit (8 byte)	0 ～ 18446744073709551615 (0xffffffffffffffff) C/C++:unsigned long long
浮動小数点		float	64 bit (8 byte)	IEEEによる
		double	64 bit (8 byte)	IEEEによる
文字列		char	8 bit (1 byte)	-128...+127 C/C++:char
よく使用されるCAPL(固有)の変数				
メッセージ	CAN	message		CANメッセージ用
タイマー	秒タイマー	Timer		秒単位のタイマー用
	ミリ秒タイマー	msTimer		ミリ秒単位のタイマー用

CAPL の変数は variables セクションに定義します。以下は定義の例です。

```
variables
{
  Message 0x100msg={DLC=5}; /*ID0x100(DLC5)の CAN メッセージを「msg」と定義*/ message
  ABSdataMSG2; /*データベースに定義されたメッセージ ABSdata を「MSG2」と定義*/


  Timer TM1; //「TM1」という名前で秒単位のタイマーを定義
  msTimer T2; //「T2」という名前でミリ秒単位のタイマーを定義
}
```

7.5 イベントハンドラー、変数、関数の使用例

次に、よく使用されるイベントハンドラーを使用した基本的な CAPL の記述例を紹介します。

7.5.1 イベントハンドラー　on key

on key は、パソコンのキーボード押下に対して反応します。

<table>
<tr><th colspan="2">使用するイベントハンドラー</th><th>on key</th></tr>
<tr><td>1.何かが起きた
（イベント発生）</td><td>2.何かを行う
（イベントに対する処理）</td><td>記述例</td></tr>
<tr><td rowspan="2">キーボードの
aが押された</td><td>出力Windowに
「‘a’が押されました」と
表示する</td><td>Variables
{
}

on key'a' //aキー押下で反応
{
write(“'a'が押されました”);
}</td></tr>
<tr><td>CANメッセージ
（ID：0x100）を送信し、
出力Windowに
「0x100を送信」と
表示する</td><td>Variables
{
message 0x100msg1={DLC=6};
//送信メッセージ0x100の定義
}

on key'a'
{
output (msg1); //aキーを押すごとに
0x100を1回送信
write (“0x100を送信”);
}</td></tr>
</table>

イベントハンドラー「on key」への定義例

```
on key ‘a’          //[a]キーを押した時に反応（大文字・小文字は区別されます）
on key ‘ ’   //スペースバーを押した時に反応
on key F1   //[F1]キーを押した時に反応
on key shiftF3    //[Shift]–[F3]キーを押した時に反応
on key PageUp   //[PageUp]キーを押した時に反応
on key Home     //[Home]キーを押した時に反応
on key *          //どれかのキーを押した時に反応
```

※ [Alt] [Esc] [Fn] [Tab] [BackSpace] キーなど、特殊用途キーは一部使用できません。

7.5.2 イベントハンドラー on message

on message は、CAN メッセージの受信に対して反応します。

<table>
<tr><th colspan="2">使用するイベントハンドラー</th><th>onmessage</th></tr>
<tr><td>1.何かが起きた
（イベント発生）</td><td>2.何かを行う
（イベントに対する処理）</td><td>記述例</td></tr>
<tr><td rowspan="2">メッセージ
0x100を
受信した</td><td>出力Windowに
「0x100を受信」と表示する</td><td>variables
{
}

on message 0x100 //ID0x100のメッセージ受信で反応
{
write (“0x100を受信”);
}</td></tr>
<tr><td>0x200を送信して出力Windowに
「0x200を送信」と表示する</td><td>variables
{
message 0x200Sendmsg={DLC=4};
//送信メッセージ0x200の定義
}

on message 0x100
{
output(Sendmsg); //0x100受信ごとに0x200を送信
write(“0x200を送信”);
}</td></tr>
</table>

イベントハンドラー「on message」への定義例

```
on message 100      //メッセージ 100(10 進数)に反応、受信チャンネルは考慮されません
on message 0x100    //メッセージ 100(16 進数)に反応、受信チャンネルは考慮されません
on message EngineData          //データベースより参照定義したメッセージ EngineData に反応
on message CAN1.0x200          //CAN1 でメッセージ 200(16 進数)を受信したら反応
on message*                    //すべてのメッセージ受信時に反応
on message CAN2.*              //CAN2 で受信したすべてのメッセージに反応
on message 100-200             //ID100 から 200(10 進数)の間のすべてのメッセージに反応
on message EngineData,ABSdata  //EngineDataABSdata どちらか受信時に反応
```

7.5.3 イベントハンドラー on start/on timer

on start は、CANoe/CANalyzer の測定開始に対して反応します。

on Timer は、タイマー設定時間経過に対して反応します。

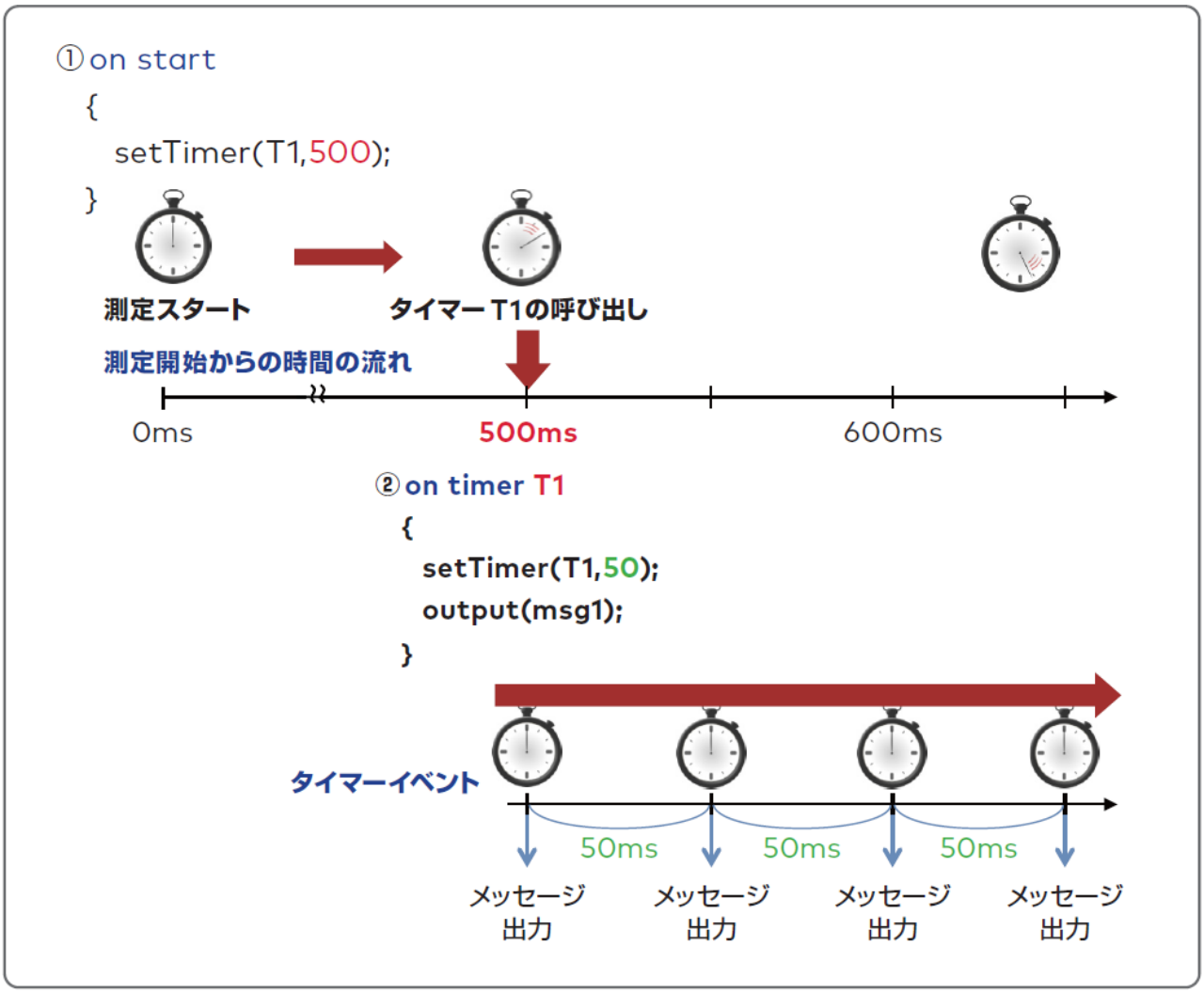
使用するイベントハンドラー		on start / on timer
1.何かが起きた （イベント発生）	2.何かを行う （イベントに対する処理）	記述例
CANoe、 CANalyzerが 測定開始された	500ms後に タイマーT1を開始し CANメッセージ （ID0x100）を 50ms周期送信する	variables { message 0x100msg1={DLC=6}; msTimer T1; } on start { setTimer(T1,500); /*測定開始500ms後に タイマー T1を呼び出す*/ } on timer T1 { setTimer(T1,50); //50ms毎T1をセット output(msg1); //メッセージを出力 }

イベントハンドラー「on timer」への定義例

CAPL は Timer の概念が 1 つのキーとなります。上の記述例を見ると、「**on start**」と「**on timer**」の両方に **CAPL 関数「setTimer」**が記述されていますが、これらはどのような役割をしているのでしょうか。また、どのようにして CAPL からメッセージの周期送信が行われるのでしょうか。これらについて、以下に解説していきます。

【Timer の特徴】

1. タイマーの開始には、setTimer() 関数を用いて時間を設定する
2. 設定時間経過後に関連するタイマーイベントハンドラー on timer が呼ばれる
3. タイマーイベントハンドラーは、タイマー経過後にイベントを実行すると同時に初期化される
4. タイマーイベントは停止条件がない限り、繰り返し処理が行われる



①で、測定開始と同時に on start が呼ばれ、タイマー"T1"がスタート

→ ② 500 ミリ秒後に on timer T1 が呼ばれ、再度タイマー"T1"をスタート。同時にメッセージを送信

→ 以降 50 ミリ秒ごとに呼ばれ、メッセージ送信を行う

このように、時間経過ごとに on timer が呼ばれてメッセージを送信することで、CAPL からの周期送信が行われています。

7.5.4 イベントハンドラー　on PDU

on PDU は、PDU の受信に対して反応します。

使用するイベントハンドラー		On PDU
1.何かが起きた （イベント発生）	2.何かを行う （イベントに対する処理）	記述例
PDU “EngineData”を 受信した	PDU “ABSdata”を送信する	variables { PDU ABSData mypdu; //送信PDUの定義 } on PDU EngineData { triggerPDU(mypdu);//EngineData受信 ごとにABSDataを送信 }

イベントハンドラー「on PDU」への定義例

```
on PDU EngineData            // EngineData に反応
on PDU short 2          // PDU shortID2（10 進数）の PDU に反応
on PDU long 2           // PDU longID2（10 進数）の PDU に反応
on PDU 2 // PDU ID2（10 進数）の PDU に反応
on PDU*  // すべての PDU 受信時に反応
```

8. 便利な CAPL 関数の紹介

CAPL関数	特徴／使用例	記述例
resetCanEx	指定したチャンネルを初期化、バスオフ時の復帰CANoe/CANalyzerがバスオフに至った場合にも、自動復帰することができます。	**on busOff** **{** **resetCanEx(1); //カッコ内は初期化するチャンネル//上の記述はCAN1を初期化** **}**
cancelTimer	本稿の第7.5.3で説明したTimerを停止します。これによりタイマーによるメッセージ周期送信をしている場合に送信を停止することができます。	**onkey'c'** **{** **cancelTimer(T1); //cキーを押下でタイマーT1を停止** **}**
getLocalTimeString	イベント発生時の時刻を取得します。 文字列のフォーマットはdddmmm dd hh:mm:ssyyyy（例: Fri Aug 21 15:22:24 2014）です。	**variables** **{** **char timeBu˜ er[64]; //時刻取得用の変数の定義** **}** **on errorFrame //エラーフレーム受信で反応** **{** **getLocalTimeString(timeBuffer);** **//イベント発生時の時間を格納** **write（"送信停止 %s",timeBuffer);** **//取得時間をwrite 関数にて** **//出力Windowに書き込む** **}**
stop	オンラインの測定を停止します。	**on key' ' //スペースキー** **{** **stop();//CANoe/CANalyzerの測定を停止** **}**

詳細は、本稿の第 11 章で紹介しているオンラインヘルプ「テクニカルリファレンス：CAPL 関数」をご参照ください。

9. キーワード“this”

CAPL の「this」とは、on message などのイベントハンドラー内で自動生成される変数です。on message で受信したメッセージのデータ（シグナル）アクセスには、「this」を使用します。 on Key では、「this」を使用することにより変数として使用できます。

例

```
variables
{
 message 0x200 Sendmsg={DLC=4};  //送信するメッセージを定義
}
on message 0x100 //メッセージ 0x100 受信時に反応
{
   Sendmsg.byte(0) = this.byte(0);  //送信メッセージの 0byte 目に 0x100 の
                                    0byte 目の値を代入
output(Sendmsg);  //0x200 を出力
}
```

```
on message EngineTemperature
                    //DB に定義されたメッセージ“EngineTemperature”受信時に反応
{
 if(this.SigTemp.phys > 100) //シグナル“SigTemp”が物理値で 100 を超えたら
 write("値が上限を超えました"); //出力 Window に“値が上限を超えました”と書き込む
}
```

```
variables
{
   PDU ABSData mypdu;  //送信する PDU を定義
}
on PDU EngineData  //DB に定義された PDU“EngineData”受信時に反応
{
   mypdu.Payload = this.Payload;   //EngineData のペイロードデータを
                                   ABSData のペイロードに代入
  triggerPDU(mypdu);   //ABSData を送信
}
```

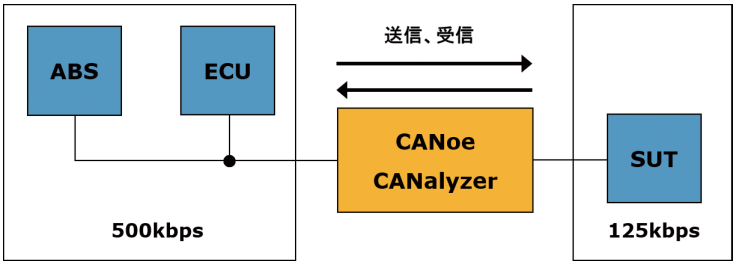
8 章、9 章で紹介した便利な CAPL 関数やキーワード「this」の詳細につきましては、「CAPL クイックガイド」をご参照ください。こちらは、ベクター・ジャパン会員専用ページ「ベクター・ジャパン Member Area」にてご覧いただけます。詳細は下記リンク先をご覧ください。（ベクターの Web サイトへ移動します）

https://www.vector.com/jp/ja/support-downloads/vj-memberarea/

10. ゲートウェイ

CAPL を用いて CANoe/CANalyzer をゲートウェイ(※)ノードとして動作させることができます。以下、2 つの CAN バスの間でメッセージを受け渡すための設定と記述例を紹介します。

※ゲートウェイとは、異なるネットワークと接続し、データを転送するためのネットワークノードのこと。



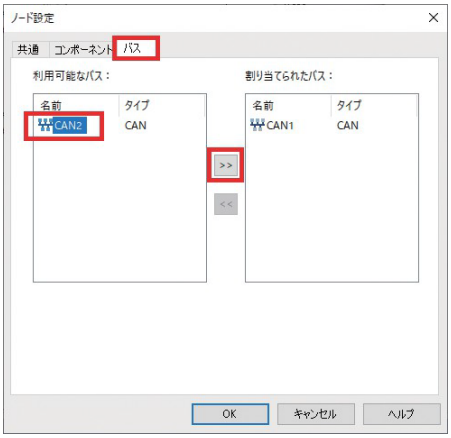
10.1 ゲートウェイノードの設定方法

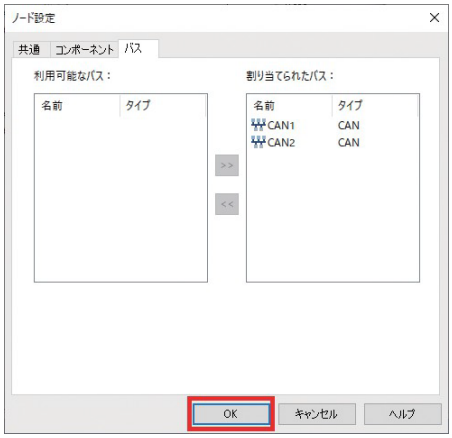
CANoe と CANalyzer の場合、ノードの設定に差異があります。本章では、4 章で示した CAPL ノードの設定は完了しており、必要な CAN チャンネルの設定は済んでいる前提で設定方法を紹介します。CAN チャンネルの詳細については、ヘルプをご確認いただくか、「はじめての CANoe/ CANalyzer」をご参照ください。

10.1.1CANoe の場合

ネットワークノードに対してゲートウェイの設定を行う必要があります。

① [ノード設定]の[バス]をクリックします。メッセージの受け渡し先のバスを設定します。ここでは [CAN2]を受け渡し先としておりますので、クリックして[＞＞]アイコンをクリックします。最後に [OK]をクリックします。





② シミュレーション設定のノードのアイコンに、[ゲートウェイ]のアイコンが追加されます。

10.1.2 CANalyzer の場合

追加の設定は不要です。

10.2 CAPL プログラム例

① 2 つのバス間で、メッセージをそのまま受け渡すための記述

```
-----------------------------------------------------------------------------

on message CAN1.*  //CAN1 でメッセージを受信
{
    message CAN2.* msg; //CAN2 から送信するメッセージの定義
    if(this.dir!=rx)return; //重要!!CAN1 から送信するメッセージに反応しないようにする記述
    msg=this;
    output(msg);
}

onmessageCAN2.*  //CAN2 でメッセージを受信
{
    message CAN1.*msg; //CAN1 から送信するメッセージの定義
    if(this.dir!=rx)return; //重要!!CAN2 から送信するメッセージに反応しないようにする記述
    msg=this;
    output(msg);
}

-----------------------------------------------------------------------------
```

② 2 つのバス間で、メッセージをそのまま受け渡すが、ある特定の ID を受信した場合に ID を変更し、データバイト値にはそのまま送信されたデータを入れ込んで送信する記述例

```
---------------------------------------------------------------------------

variables
{
  message CAN2.0xFFGWmsg={DLC=8};  //送信するメッセージ ID および DLC の定義 const
  dwordtargetID=0xc9;  //特定 ID を定義
}
onmessageCAN1.*  //CAN1 でメッセージを受信
{
  message CAN2.*msg;      //CAN2 から送信するメッセージの定義
  if(this.dir==rx) //CAN1 のメッセージが受信の場合のみ処理を実行する
  {
    if(this.id==targetID) //ターゲットとなる特定 ID の場合
    {
      GWmsg.byte(0)=this.byte(0); //各バイトへデータ値の入れ込み
      GWmsg.byte(1)=this.byte(1); //ここでは 8 バイト分を行っているが
      GWmsg.byte(2)=this.byte(2); //実データ長に合わせ削除等を行う
      GWmsg.byte(3)=this.byte(3);
      GWmsg.byte(4)=this.byte(4);
      GWmsg.byte(5)=this.byte(5);
      GWmsg.byte(6)=this.byte(6);
      GWmsg.byte(7)=this.byte(7);
      output(GWmsg);
    }
    else //特定の ID 以外はそのまま転送
    {
      msg=this;
      output(msg);
    }
  }
}

---------------------------------------------------------------------------
```

※CANoe の場合、CAPL 関数 SetBusContext()を使用して、測定実行時に動的に転送先のチャンネルを設定することもできます。

11. CAPL 関数一覧

CAPL 関数一覧（リファレンス）は、CANoe/CANalyzer のオンラインヘルプにあります。

【オンラインヘルプ起動方法】

CANoe/CANalyzer を起動し、メニューバー から [ヘルプ] を選択

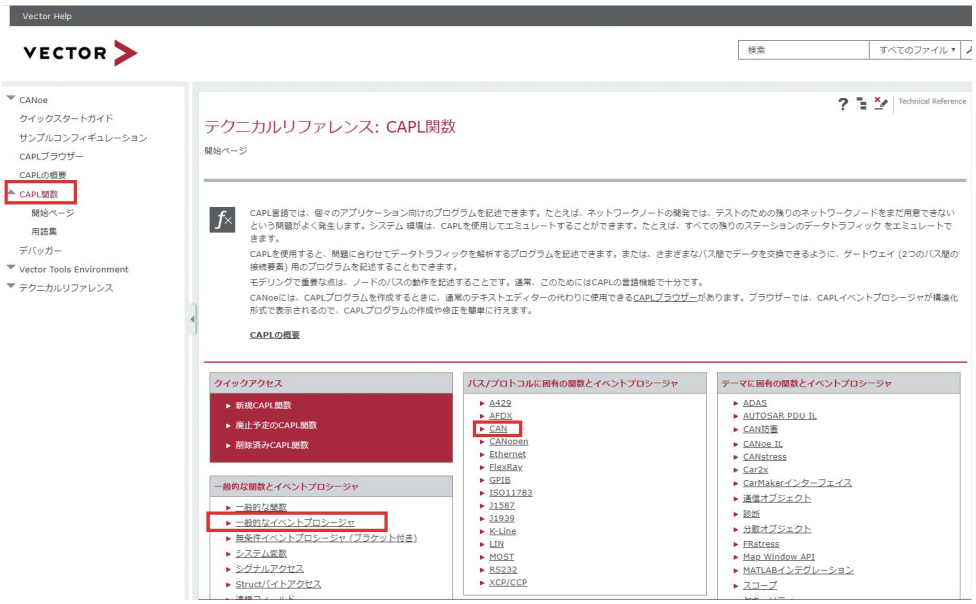
オンラインヘルプの起動後、画面左側のツリーから「CAPL 関数」を選択してください。

・**on timer / on PDU** など、プロトコルに関連しないイベントハンドラー

→「一般的なイベントプロシージャ」

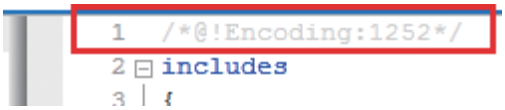
・**on message / output()**など、CAN に関連する関数、イベントハンドラー

→「バス/プロトコルに固有の関数とイベントプロシージャ」→「CAN」



また、CANoe/CANalyzer 付属のサンプルコンフィギュレーションでは、CAPL の使用例を確認できます。サンプルコンフィギュレーションの起動方法については、「はじめての CANoe」もしくは「はじめての CANalyzer」を参照してください。

※付属のサンプルコンフィギュレーションのソースコードを流用してプログラムを作成する場合などは、コンパイル時に警告が表示される場合があります。こちらはお使いの環境のエンコーディング設定に起因するものです。警告のため無視して構いませんが、ソースコードの先頭行のエンコーディング定義を変更して解決できます。
例）お使いの Windows 環境が日本語の場合、下記「1252」部分を「932」に変更します。
（1252 は Western European (Windows)、932 は Japanese (Shift-JIS)を表します。）



12.その他のCAPL便利機能

CAPL に関する便利機能を紹介します。

12.1 インクルードファイル

他の CAPL プログラムで使用している定数、変数および関数を取り込み（インクルード）、プログラムを作成できます。これにより、既存の CAPL プログラムで使用している定数などを再利用できるため、プログラム作成の手間を削減することができます。詳細は、オンラインヘルプ「インクルードファイル：概要」をご参照ください。

12.2CAPL 暗号化

CAPL ブラウザーでは作成した CAPL ファイルを暗号化して保存できます。暗号化して保存されたファイルは編集／閲覧ができなくなり、実行のみが可能となります。セキュリティの計算式など、秘匿したい情報が含まれた CAPL ファイルと CANoe/CANalyzer コンフィギュレーションファイルを配布する際に有効となります。詳細は、オンラインヘルプ「暗号化して保存」をご参照ください。

12.3 デバッガー機能（CANoe のみ）

測定中に作成したプログラムの機能を解析することができます。ソースコードの任意の箇所にブレークポイントを設定してその時の内部値を確認したり、期待した分岐制御が実施されているかを確認できます。詳細は、オンラインヘルプ「デバッガー」をご参照ください。

13.参考（文法）

ご参考に、一般的な C 言語等のプログラミングに充当する内容に関して紹介します。

13.1 演算子について

演算子は、C 言語と同じものを使用することができます。簡単なものをいくつか紹介します。

① 算術演算子

加算「+」、減算「-」、乗算「*」、除算「/」、余り「%」を求めます。

② 比較演算子

等しい値「==」、等しくない値「!=」、不等号「>」「>=」「<=」「<」を調べます。

演算子	使用例	説明
==	a == b	aとbは等しい
!=	a != b	aとbは等しくない
>	a > b	aはbより大きい
>=	a >= b	aはb以上

③ 論理演算子

真と偽を論理演算で評価します。

演算子	使用例	説明
&&	a && b	AND（かつ） 戻り値: TRUEまたはFALSE
||	a || b	OR（または） 戻り値: TRUEまたはFALSE
!	! a	NOT（否定） TRUEをFALSEへ、FALSEをTRUEへ変更

④　インクリメント／デクリメント演算子

変数を 1 加算「++」、変数を 1 減算「--」します。

⑤　ビット演算子

各変数をビット単位で処理（計算）するための演算子で、以下のようなものがあります。

演算子	意味	使用例	説明
&	ビット積 （AND）	a=b&c;	bとcの各ビットの論理積の結果をaに代入 a = 1 & 7 ← 結果aは1 (0001 & 0111 ⇒ 0001)
|	ビット和 （OR）	a=b|c;	bとcの各ビットの論理和の結果をaに代入 a = 1 | 7 ← 結果aは7 (0001 | 0111 ⇒ 0111)
~	ビット否定 （NOT）	a=~b;	bの各ビットを反転しその結果をaに代入 a = ~1 ← 結果aはbyte型で254（00000001 ⇒ 11111110）

⑥　シフト演算子

2 進数の各ビットを右または左にずらす演算子です。

演算子	意味	使用例	説明
>>	右にシフト	a=b>>c;	bを右にcビットずらす
<<	左にシフト	a=b<<c;	bを左にcビットずらす 1 << 3 ← 結果は 8 (0001 ⇒ 1000)

13.2 制御文

CAPL は、C 言語と同じように、制御文を使用することができます。また、反復処理から脱出するための「break 文」や、反復処理をスキップするための「continue 文」も使用できます。

次に、ステートメント（命令文）の説明と基本例文を記します。

ステートメント	説明	一般的な形式
IF文	if文を使用して式を評価し、評価結果が「真(True)」の場合のみ命令が実行されます。評価結果が「偽(False)」の場合、コードブロックは実行されません。 注記：if 文はネストする (=入れ子構造にする)ことが可能です。例えばif 文の中にさらに (1つ以上の) if文を記述することができます。	if (条件式) { .. 条件式の結果がTrueの場合に実行される命令文.. } **例**: if (Blinker_On == 1) { write (“Blinker is on.”); }
IF-ELSE文	if-else文も式を評価できます。評価した式の結果によって、2つの命令のいずれかが実行されるという点がif 文と異なります。式の結果が「真 (True)」であればif節の命令が実行され、結果が「偽 (False)」であれば、else 節の命令が実行されます。 注記：if-else文も、if 文同様にネストする (=入れ子構造にする)ことができ、たとえばif 節の中にさらにif-else文を、else 節の中にさらにif- else文を記述することができます。	一般的な形式： if (条件式) { ...条件式の結果がTrueの時に実行される命令文... } else { ...条件式の結果がFalseの時に実行される命令文... } **例**: //時計の「分」を進める関数 increment_clock_minutes() { int clock_minutes; if (clock_minutes >= 60) clock_minutes = 0; else clock_minutes++; }
SWITCH文	switch文は、if-else文と同じ制御になりますが、2つ以上の選択肢を必要とする場合に使用すると、より便利です。switch文の中にcase文を複数記述することによって1つの文に2つ以上の分岐を作ることができます。 switch文は、式の「値」に応じて実行内容を分岐します。「値」と「整数または文字の定数」を比較し、一致した選択肢の文を実行します。つまり、1つ以上のcase文と比較し、{}内で最初に一致した文の制御が実行されます。case文はいくつあってもよく、1つの{}の中にすべて記述できます。 caseの後ろは、整数の定数、文字の定数、定数式のいずれかになり、定数式には、変数や関数を置くことはできません。 注記：どのcase文にも値が一致しない場合、default文が実行されます。default文がない場合(defaultは省略も可能)、witch文を抜けます。	**例**: float value1, value2, result; char operator; ... switch (operator) { case ‘+’ : result = value1 + value2; break; case ‘-’ : result = value1 - value2; break; case ‘*’ : result = value1 * value2; break; case ‘/’ : result = value1 / value2; break; }

ステートメント	説明	一般的な形式
WHILE ループ	while文では、その条件が成立している間(=比較している式が「偽 (False)」になるまで)同じ処理が繰り返し行われます。 注記：命令文が実行される前に、必ず最初に条件式が評価されます。結果が「偽(False)」の場合には、whileループ内の命令文は実行されません。	while (条件式) { ..条件式の結果がTrueの場合に実行される命令文.. } **例**: //カウントが10になるまでループする while (count < 10) { count ++; }
DO-WHILE ループ	do-while文は、while文と同じ制御ですが、実行中に少なくとも1度はループ内の命令を実行する必要がある場合に用いると、より便利です。 このループ文は、最初に命令文を実行し、次に条件式を評価します。条件式が「真(True)」の場合、それが「偽 (False)」になるまで命令文の始めから繰り返し実行 (ループ) されます。	Do { ...条件式の結果がTrueの場合に実行される命令文... } while (条件式); **例**: //カウントが10になるまでループする do { count ++; } while (count != 10);
FORループ	for文もまた、while、do-whileと同様に繰り返し実行されるループ文の1つです。定められた回数だけ繰り返し実行する場合に用いると、便利です。 注記： 「初期設定式」とは、最初にfor文が処理される時に1度だけ実行される式です「。条件式」の意味はwhile文のものと同じで「真（true）」の時にループします。「再設定式」は、“繰り返す処理”が最後まで終了した時に実行される式で、ループするたび（「繰り返す処理」が終了するたび）に実行されます。	for (初期設定式; 条件式; 再設定式) { ...条件式の結果がTrueの場合に実行される命令文... } **例**: //10回ループする int i, count; for (i = 0; i < 10; i ++) { count ++; } **注意**:条件式を省略した場合、for文は無限ループになりますのでご注意ください。またCAPLではWaitのためのfor文は使用できません。

ステートメント	説明	一般的な形式
BREAK文	break文はswitch、do-while、while、forの繰り返し文の中でループから抜け出すために用います。	例はSWITCH文参照
CONTINUE文	continue文は、switch、do-while、while、forの繰り返し文の中で、ループ中の残りの処理をスキップして、ループの先頭に戻ります。	**例:** int x; while (x != 0) { doThis (x); if(x == 4) continue; doThat (x); }
RETURN文	return文はユーザー定義の関数で、イベントハンドラーや他のユーザー定義の関数に値や式の値を返す場合に使用します。return文は、ループの中に引数がない場合、ループを抜けることもできます。	例: long Power(int x, int y) { int i; long result; result = 1; for (i = 1; i <= y; i++) { result *= y; } return result; }

14. お問い合わせ窓口

技術関連のお問い合わせは、ベクターサポートまでお寄せください。

✓ サポートリクエスト（Web サイト経由のサポート問い合わせフォーム）
www.vector.com/jp/ja/support-downloads/support/

【お問い合わせ時のお願い】

お問い合わせの際、サポート開始前に製品のバージョン（例：CANape バージョン xx.x など）、シリアルナンバーなどをお伺いいたします。お手数ではございますが、事前にご確認のうえ、お問い合わせくださいますようお願い申し上げます。

【サポート対応について】

夜間／休日に頂いたお問い合わせは翌営業日の対応となります。また、内容によりご回答までにお時間を頂く場合がございます。なお、不具合が発生した際にご使用されていた関連プログラム一式のコピーを、検証のためご提供いただくようお願いする場合がございます。何卒ご理解ご協力のほど、よろしくお願い申し上げます。

✓ **お見積もり／保守契約関連のお問い合わせ**

営業部　TEL：（東京）03-4586-1808　E-mail：sales@jp.vector.com

✓ **トレーニングに関するお問い合わせ**

トレーニング部　TEL：（東京）03-4586-1816　E-mail：training@jp.vector.com

【ベクターのトレーニング】

ベクターでは、車載およびオープンネットワークに関連する技術や、その開発ツールの使用法など、エンジニアの皆さまに向けた専門トレーニングをご用意しています。トレーニングは、東京本社および名古屋支社にて行っております。受講のお申し込みについては下記 URL をご覧ください。定員になり次第締め切らせていただきますので、お早めにお申し込みください。

また、規模やご要望に応じて、お客様ご指定の場所でのオンサイト・トレーニングも実施しております。詳細は、ベクターのトレーニング部までお気軽にお問い合わせください。

https://academy.vector.com/jp/ja/

VECTOR >

MORE INFORMATION
Visit our website for:

> News
> Products
> Demo software
> Support
> Training and workshops
> Contact addresses

vector.com