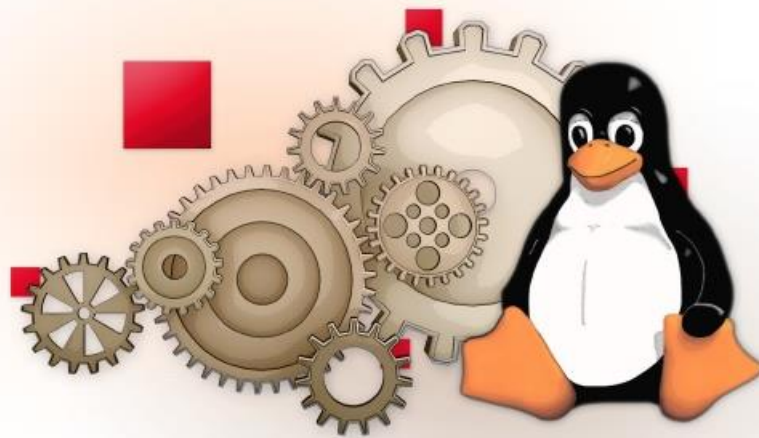


PCAN Driver for Linux v8

CAN Driver and Library API for Linux

User Manual



関連商品

Product name	Version	Part number
PCAN Driver for Linux	8.x.x	N/A (該当なし)

PCAN®は、PEAK-System Technik GmbH の登録商標です。

本書に記載されているその他の製品名は、それぞれの会社の商標または登録商標である可能性があります。それらは ™ または ® で明示的にマークされていません。

©2021 PEAK-System Technik GmbH

このドキュメントの複製（コピー、印刷、またはその他のフォーム）および電子配布は、PEAK-System Technik GmbH の明示的な許可がある場合にのみ許可されます。PEAK-System Technik GmbH は、事前の発表なしに技術データを変更する権利を留保します。一般的なビジネス条件とライセンス契約の規制が適用されます。すべての権利は留保されています。

PEAK-System Technik GmbH

Otto-Roehm-Strasse 69

64293 Darmstadt

Germany

Phone: +49 6151 8173-20

Fax: +49 6151 8173-29

www.peak-system.com

info@peak-system.com

Document version 3.6.1 (2021-01-28)

目次

1 免責事項.....	4
2 はじめに.....	5
2.1 特徴.....	5
2.2 システム要件.....	6
2.3 納品範囲.....	6
3 インストール.....	7
3.1 バイナリのビルド.....	7
3.2 パッケージのインストール.....	9
3.3 ソフトウェアのコンフィグレーション.....	9
3.4 Non-PnP ハードウェアの構成.....	12
4 ドライバの使用法.....	14
4.1 ドライバの読み込み.....	14
4.2 Udev ルール.....	16
4.3 / proc インターフェイス.....	19
4.4 / sysfs インターフェイス.....	21
4.5 lspcan ツール.....	24
4.6 read / write インターフェイス.....	26
4.7 test ディレクトリ.....	28
4.7.1 receivetest.....	29
4.7.2 transmitest.....	30
4.7.3 pcan-settings.....	31
4.7.4 bitratetest.....	32
4.7.5 pcanfdtst.....	33
4.8 netdev モード.....	38
4.8.1 パラメータの割り当て.....	39
4.8.2 defclk パラメータ.....	39
4.8.3 ifconfig / iproute2.....	40
4.8.4 can-utils.....	42
4.9 USB 大容量ストレージデバイスモード.....	42
5 開発者ガイド.....	46
5.1 chardev モード.....	46
5.1.1 CAN 2.0 API.....	47
5.1.2 CAN FD API.....	51
5.2 netdev モード.....	65

1 免責事項

提供されるファイルは、PCAN Driver for Linux パッケージの一部です。

これはフリーソフトウェアです。Free Software Foundation によって発行された GNU General Public License の条件の下で、それを再配布および変更することができます。ライセンスのバージョン 3、または (オプションで) 以降のバージョンのいずれか。

このソフトウェアは配布していますが、保証はありません。商品性または特定目的への適合性に関する黙示の保証もありません。詳細については、GNU General Public License を参照してください。

ソフトウェア パッケージと一緒に GNU General Public License のコピーを入れております。そうでない場合は、<https://www.gnu.org/licenses/> を参照してください。




重要な注意： 提供されたソース コードの知的財産を、互換性のあるハードウェアの開発または製造に使用することは固く禁じられています。すべての権利は、PEAK-System Technik GmbH に留保しています。

2 はじめに

Linux 用の PCAN ドライバを使用すると、Linux ベースのシステムで CAN 2.0 を使用でき、PCAN ドライバ v8 以降で、PEAK-System の CAN FD ハードウェア製品を使用できます。Linux 2.4 カーネルの使用が減少している場合でも、このカーネルラインは幾つかのバージョンおよび古い PEAK-System ハードウェア製品との互換性は保証されています。

このドライバは、一般的な “Real Time Driver Mode” モデルに接続することにより、Xenomai¹ や RTAI² などのリアルタイム (RT) 拡張機能の最新バージョンとも互換性があります。

これまで、Linux 用の PCAN ドライバは、character モードのデバイスドライバシステムコール (open、read、write、close、poll、ioctl) を実装することにより、*chardev* というアプリケーションプログラミングインターフェイスを提供します。バージョン 20070306_n 以降、ドライバは *netdev* インターフェイスも提供しています。これは、Kernel SocketCAN ネットワークサブレイヤを統合することにより、アプリケーションが Linux カーネルのソケットインターフェイスを介して PEAK-System の CAN チャンネルにアクセスできるようになりました。インターフェイスの選択は、ドライバのビルド時に排他的に行われます。ドライバは、両方のインターフェイスを同時に提供して実行することはできません。

 **注:** Linux カーネル v3.6 以降、PEAK-System は、最も使用されている PC CAN インターフェイスのサポートをメインラインカーネルに組み込むように取り組んでいます。ソケットベースのアプリケーションから PEAK-System が作成した PC CAN インターフェイスを使用して CAN バスにアクセスする場合は、Linux 用 PCAN Driver for Linux package をインストールする必要はありません。ただし、*netdev* インターフェイスは、下位互換性のために残されています。

Linux 用 PCAN ドライババージョン 8 は、CAN FD 仕様をサポートしたため、大きな進化を遂げました。CAN FD をサポートするために、古い *chardev* API も進化する必要がありました。古い *chardev* API もサポートしています。

PEAK-System によって作成した新しいハードウェア製品のサポート、ツールとカーネルの新しいバージョン、またはいくつかのバグ修正のために、パッケージは常に進化しています。最新バージョンは、PEAK-System の Web サイトからダウンロードできます。

<https://www.peak-system.com/linux/>

2.1 特徴

- PEAK-System 製のすべての CAN2.0 a / b および CAN FD ハードウェア製品をサポート
 - 32 および 64 ビット環境で Linux カーネル 2.6.x、3.x、および 4.x をサポート
 - DESTDIR とクロスコンパイルをサポート
 - Udev システムのサポート
 - 拡張された *sysfs* 統合
 - 最適化された文字モード デバイスドライバインターフェイス (*chardev*) は、CAN 2.0 および CAN FD 標準およびアプリケーションとドライバ間の multiple messages transfers をサポート
 - 強化された NETLINK 統合 (*ip link* のサポート)、SocketCAN デバイスドライバインターフェイス (*netdev*) は CAN2.0 および CAN FD をサポート
-
- ¹ ウェブサイト Xenomai: <https://xenomai.org>
 - ² ウェブサイト RTAI: <https://www.rtai.org>

- Xenomai 3.x、RTAI 4.x、5.x などのリアルタイムの Linux 拡張機能は、ドライバ、ユーザースペースライブラリ、テストおよびサンプル アプリケーション (*chardev* インターフェイスのみ) でサポートされています。
- 古いバージョンのドライバ (7.x 以前) で実行される既存の CAN 2.0 *chardev* アプリケーションとは完全なバイナリ互換

2.2 システム要件

- 32 または 64 ビットのカーネルを実行する Linux ベースのシステム
- PEAK-System の PC CAN インターフェイス
- make、gcc
- 実行中の Linux のカーネルヘッダー (または Linux ヘッダー) パッケージ、またはクロスコンパイルされたカーネルのソースツリー
- g++ および libstdc++
- libpopt-dev パッケージ

 **注:** g++ コンパイラと libpopt-dev パッケージは、test ディレクトリからユーザースペースアプリケーションをビルドする場合にのみ必要です。

2.3 納品範囲

- Linux 用 PCAN ドライバのインストール
 - デバイスドライバモジュールソースと Makefile
 - ユーザーライブラリソースと Makefile
 - “test” およびツールのアプリケーションのソースと Makefile
 - Udev ルール
- PDF フォーマットのドキュメント (このユーザー マニュアル)

3 インストール

Linux 用の PCAN ドライバは**ツリー外**のドライバモジュールであり、GPL のため、ドライバのソースファイル、ユーザーライブラリ、一部のテストユーティリティとツールを含む [peak-linux-driver-8.12.0.tar.gz](#) ファイル (*tarball* ファイル) で提供されます。(6 ページの 2.3 納品範囲を参照)。

この章では、non-RT および RT Linux システムでのドライバパッケージ全体のセットアップについて説明します (インストール部分には root 権限が必要です)。また、クロスコンパイルオプションについても説明します。

3.1 バイナリのビルド

▶ パッケージをインストールするには、次の手順を実行します：

1. \$ HOME (たとえば) ディレクトリから圧縮された [peak-linux-driver-8.12.0.tar.gz](#) ファイル (*tarball* ファイル) を解凍します：


```
$ tar -xzf peak-linux-driver-X.Y.Z.tar.gz
$ cd peak-linux-driver-X.Y.Z
```

2. 全てを消去する：

```
$ make clean
```

▶ デフォルト設定で non-real time (ノンリアルタイム) バイナリを構築するには：

```
$ make
```

 **注：**この動作は、ドライバ v8.x からの新機能です。以前のバージョンでは、グローバル make コマンドは、*chardev* インターフェイスではなく *netdev* インターフェイスを有効にしてビルドしていました。この変更の主な理由は、多数の PEAK-System CAN ハードウェア製品が、SocketCAN インターフェイスとしてメインラインカーネルによってネイティブにサポートされるようになったためです³。ユーザードライバは、*netdev* の代わりに *chardev* インターフェイスを使用することができるようになりました。*netdev* インターフェイスは、以下を使用してドライバ (のみ) を再構築することで使用できます。

```
$ make -C driver NET=NETDEV_SUPPORT
```

▶ Xenomai カーネルで実行されるリアルタイムバイナリを構築するには：

```
$ make RT=XENOMAI
```

 **注：**ドライババージョン 8.2 以降、Xenomai バイナリを次の方法でビルドすることもできます。

```
$ make xeno
```

³ Kernel code: https://elixir.bootlin.com/linux/v3.4/source/drivers/net/can/usb/peak_usb/pcan_usb_core.c

▶ RTAI カーネルで実行されるリアルタイムバイナリを構築するには：

```
$ make RT=RTAI
```

i 注：ドライババージョン 8.2 以降、次のコマンドを使用して RTAI バイナリをビルドすることもできます。

```
$ make rtai
```

i 注：上記のリアルタイムコンパイルの 1 つを選択すると、non-RT PC 用 CAN インターフェイス（USB アダプターなど）の一部のサポートが削除されます。

▶ バイナリをクロスコンパイルするには：

```
$ make KERNEL_LOCATION=/where/are/the/kernel/headers
```

パッケージ (peak-linux-driver-8.12.0.tar.gz) を解凍すると、次のようになります。

1. driver ディレクトリ、
2. lib ディレクトリ、および libpcanbasic ディレクトリ
3. test ディレクトリ。

これは、次の 3 つのコマンドと同等です。

```
$ make -C driver
$ make -C lib
$ make -C test
```

▶ ライブラリの 32 ビットバージョンの作成：

ドライババージョン 8.5 以降、現在の C コンパイラで実行する場合、64 ビットカーネルを実行すると、32 ビットバージョンの libpcan ライブラリが自動的にビルド（およびインストール）されます。

i 注：gcc-multilib パッケージをインストールする必要があります。

non-RT 構成の Linux 用 PCAN ドライバのデフォルト構成は、すべての PC 用 CAN インターフェイスのサポートを処理します。ただし、メモリを節約したり、クロスコンパイルやロードの問題を修正したりするために、これらのインターフェイスの一部のサポートを削除することができます。ドライバの Makefile は、make コマンドラインから次の一連のスイッチを処理します。

Variable	Value	Description
DNG	DONGLE_SUPPORT	PEAK-System からの CAN インターフェイスの平行ポートのサポートをドライバに含めます (デフォルト)
	NO_DONGLE_SUPPORT	CAN インターフェイスの平行ポートのサポートをドライバから削除します
USB	USB_SUPPORT	PEAK-System からの CAN インターフェイスの USB のサポートをドライバに含めます (デフォルト)
	NO_USB_SUPPORT	ドライバから CAN インターフェイスの USB のサポートを削除します
PCI	PCI_SUPPORT	PEAK-System からの CAN インターフェイスの PCI / PCIe のサポートをドライバに含めます (デフォルト)
	NO_PCI_SUPPORT	ドライバから PCI / PCIe CAN インターフェイスのサポートを削除します
PCIEC	PCIEC_SUPPORT	PEAK-System からの CAN インターフェイスの ExpressCard のサポートをドライバに含めます (デフォルト)
	NO_PCIEC_SUPPORT	ExpressCardCAN インターフェイスのサポートをドライバから削除します

Variable	Value	Description
ISA	ISA_SUPPORT	PEAK-System からの CAN インターフェイスの ISA / PC104 のサポートをドライバに含めます (デフォルト)
	NO_ISA_SUPPORT	ISA / PC104CAN インターフェイスのサポートをドライバから削除します
PCC	PCCARD_SUPPORT	PEAK-System からの CAN インターフェイスの PCCard のサポートをドライバに含めます (デフォルト)
	NO_PCCARD_SUPPORT	ドライバから CAN インターフェイスの PCCard のサポートを削除します

表 1 : サポートされている PC CAN インターフェイススイッチ

たとえば、PCAN-Dongle も PCAN-PC カードの CAN インターフェイスもサポートせずにドライバをビルドするには、次のようにします。

```
$ make -C driver DNG=NO_DONGLE_SUPPORT PCC=NO_PCCARD_SUPPORT
```

3.2 パッケージのインストール

▶ バイナリが構築されたら、次の手順を実行してパッケージをインストールします :

1. ドライバパッケージのルートディレクトリにあることを確認してください :

```
$ cd peak-linux-driver-X.Y.Z
```

2. すべてをインストールします (root 権限が必要です) :

a) Debian-based のシステムでは、ユーザーは `sudo` コマンドを使用できます :

```
$ sudo make install
```

b) それ以外の場合、インストールは次の方法で行われます :

```
$ su -c "make install"
```

上記のセットアップでは、実行中のシステムにドライバ、ユーザーライブラリ、およびテストプログラムをビルドしてインストールします。

3.3 ソフトウェアのコンフィグレーション

Linux 用の PCAN ドライバは、いくつかのデフォルト設定で実行されます。それらのいくつかは、モジュールがロードされたときにモジュールにパラメータを渡すことによって変更できます。

Parameter	Type									
type	", " (カンマ) で区切られた文字列のリスト。	Plug-and-Play で検出できない (最大) 8 つの PC CAN インターフェイスのリストを示します。既知のタイプは次のとおりです :								
		<table border="1"> <thead> <tr> <th>type</th> <th>PC CAN interface</th> </tr> </thead> <tbody> <tr> <td>isa</td> <td>ISA および PC/104</td> </tr> <tr> <td>sp</td> <td>Standard parallel port</td> </tr> <tr> <td>epp</td> <td>Enhanced parallel port</td> </tr> </tbody> </table>	type	PC CAN interface	isa	ISA および PC/104	sp	Standard parallel port	epp	Enhanced parallel port
		type	PC CAN interface							
		isa	ISA および PC/104							
		sp	Standard parallel port							
epp	Enhanced parallel port									

Parameter	Type	
io	"," (カンマ) で区切られた 16 進数値のリスト。	対応する PC CAN インターフェイスとのダイアログに接続するための I/O ポートのリストを提供します (type を参照)。
irq	"," (カンマ) で区切られた 10 進数値のリスト。	対応する PC CAN インターフェイスとのダイアログに接続するための IRQ レベルのリストを提供します (type を参照)。
btr0btr1	16 進数値	すべての CAN/CAN FD チャンネルに設定されているデフォルト (nominal) ビットレート値を変更します。16 進数値は BTR0BTR1 値として解釈されます (SJA1000 仕様を参照)。モジュールのロード時にこのパラメータが指定されていない場合、デフォルトのビットレート値は 0x1c (500 Kbit/s) です。
bitrate	数値。末尾の k は係数 1,000 として解釈され、末尾の M は係数 1,000,000 として解釈されます。	すべての CAN/CAN FD チャンネルに設定されているデフォルト (nominal) ビットレート値を変更します。モジュールのロード時にこのパラメータが指定されていない場合、デフォルトのビットレート値は 0x1c (500 kbit/s) です。以下の注記も参照してください。
dbitrate	数値。末尾の k は係数 1,000 として解釈され、末尾の M は係数 1,000,000 として解釈されます。	オープン時にすべての CAN FD チャンネルに設定されているデフォルトのデータ ビットレート値を変更します。モジュールのロード時にこのパラメータが指定されていない場合、デフォルトのデータ ビットレート値は 2,000,000 (2 Mbit/s) です。 v8.11 以降、dbitrate が 0 の場合、CAN-FD デバイスは CAN 2.0 A/B モードでのみ初期化されます。

Parameter	Type	
assign	文字列	PCAN と SocketCAN レイヤ間のデフォルトの名前割り当てを変更します (4.8.1 パラメータの割り当てを参照)。このパラメータは、netdev インターフェイスが選択されている場合にのみ使用されます。
usemsi	数値	このパラメータは、PCIe ベースの CAN 2.0 カードの MSI の使用を制御します：
		0 INTA モード (no MSI)
		1 Full MSI モード (チャンネル毎に 1 つの IRQ)
		2 Shared MSI (デバイス毎に 1 つの IRQ)
		0 がデフォルトのモードです。
fdusemsi	数値	このパラメータは、PCIe ベースの CAN FD カードの MSI の使用を制御します：
		0 INTA モード (no MSI)
		1 Full MSI モード (チャンネル毎に 1 つの IRQ)
		2 Shared MSI (デバイス毎に 1 つの IRQ)
		0 がデフォルトのモードです。
rxqsize	数値	ドライバがチャンネル Rx キューに保存できるメッセージの最大数を定義します。Rx キューがいっぱいになると、次の着信 CAN フレームはドライバによって破壊され、CAN_ERR_OVERRUN (0x0002) フラグが設定されます。デフォルト値は 500 です。

Parameter	Type	
txqsize	数値	アプリケーションがチャンネル Tx キューに保存できるメッセージの最大数を定義します。Tx キューがいっぱいになると、次の書き込みでアプリケーションがブロックされるか、O_NONBLOCK が設定されている場合は errno = EAGAIN で失敗します。デフォルト値は 500 です。
deftsmode	数値	<p>PC CAN インターフェイスがタイムスタンプを提供できる場合、ドライバによるハードウェアタイムスタンプの処理方法を制御します：</p> <ul style="list-style-type: none"> 0 タイムスタンプは、Host (ソフトウェア) のタイムスタンプです。受信した CAN フレームのタイムスタンプは、フレームがドライバの Rx キューに保存された時間に対応します。 1 タイムスタンプは、Host タイム + ハードウェアタイムスタンプからなるオフセットに基づいています。Host タイム ベースは、PC CAN インターフェイスから通知を受信すると、ドライバによって定期的に更新されます。オフセットは、受信した CAN フレームで見つかったハードウェアタイムに対応します。 2 1 と同じですが、ハードウェアオフセットは、CPU と PC CAN インターフェイス・クォーツの間で発生する可能性のあるクロック・ドリフトを処理するように調整されている点が異なります。 3 タイムスタンプは、PCAN インターフェイスから受信したハードウェアタイムスタンプで構成されています。これは、このタイムスタンプがホスト時間ではなく、s と μs のカウントであることを意味します。PC CAN インターフェイスが初期化されているためです。 4 予約済み 5 タイムスタンプ測定が EOF ではなく SOF でトリガーされることを除いて、1 と同じです (デバイスが許可する場合)。 6 タイムスタンプ測定が EOF ではなく SOF でトリガーされることを除いて、2 と同じです (デバイスが許可する場合)。 7 タイムスタンプ測定が EOF ではなく SOF でトリガーされることを除いて、3 と同じです (デバイスが許可する場合)。 <p>デフォルト モードは PC CAN インターフェイスに依存します：</p> <ul style="list-style-type: none"> 0 SJA1000 ベースの内部バス PCAN インターフェイスのデフォルトおよびユニークモード。 1 PCAN-USB のデフォルトモード。 2 CAN FD PCAN インターフェイス
defclk	文字列	ドライバが netdev モードでビルドされている場合は、PCAN チャンネルのデフォルトのクロック値を定義します (39 ページの 4.8.2 defclk パラメータを参照)。
drvclkrf	数値	<p>ドライバがアプリケーションに提供するホストのタイムスタンプを計算するために、ドライバが基づいているクロックリファレンスを定義します。</p> <p>有効な値は次のとおりです：</p> <ul style="list-style-type: none"> 0 Real-time clock 1 Monotonic clock 4 Monotonic raw clock 7 Boot time clock <p>21 ページの「表 6: ドライバがタイムスタンプに使用するクロックリファレンス」も参照してください。</p>

表 2：ドライバモジュールのパラメータ

i **注:** ドライバの v8.x 以降、`bitrate = parameter` が変更されました。以前のバージョンでは、このパラメータでデフォルトの nominal ビットレートを変更できましたが、SJA1000 BTR0BTR1 レジスタのコーディングフォーマットにのみ従うだけでした。

既存のコンフィグレーションとの下位互換性を確保するために、`bitrate = parameter` が次のように解析されるようになりました。

- 指定された値の最初の 2 文字が `0x` または `0X` であり、16 進値が 65536 より小さい場合、値は常に BTR0BTR1 ビットレート仕様として解釈されます（ドライバが以前のバージョンで行ったように）。
- それ以外の場合、値が明らかに数値である場合は、ビット/秒 (bit / s) ビットレート仕様として使用されます。

これらのパラメータとその値は、`insmod` コマンドラインで指定するか、`/etc/modprobe.d/pcan.conf` ファイルに書込むことができます。システム管理者は、このファイルを編集してから、`options pcan` 行のコメントを解除し、独自の設定を書き込む必要があります。

3.4 Non-PnP ハードウェアの構成

i **注:** この段落は、一部の non-plug-and-play PC CAN インターフェイス (PCAN-ISA および PC / 104 PC CAN インターフェイスファミリなど) のユーザーにのみ関係します。PCI / PCIe および USB CAN インターフェイスファミリのドライバのコンフィグレーションは、システムによって処理されます。

一部の non-plug-and-play PC CAN インターフェイスを使用する場合、これらのボード用に構成された IRQ および I/O ポートをドライバに通知する必要があります (提供されているハードウェアリファレンスおよび対応するジャンパーの使用法を参照)。Linux 用の PCAN ドライバのインストール手順では、ドライバがロードされたときにドライバに渡されるいくつかのオプションの引数を定義できるコンフィグレーションテキストファイルが既に作成されています (3.3 ソフトウェアの構成ページ 9 を参照)。

たとえば、Linux ホストに 2 チャンネルの ISA PC CAN インターフェイスボードが装備されている場合、また、IRQ 5 (または IRQ 10) および I/O ポート `0x300` (または `0x320`) が、ボード上の専用ジャンパーによって選択された構成である場合、`/etc/modprobe.d/pcan.conf` ファイルを次のように変更する必要があります。

```
$ sudo vi /etc/modprobe.d/pcan.conf
# PCAN - automatic made entry, begin -----
# if required add options and remove comment
options pcan type=isa,isa irq=10,5 io=0x300,0x320
install pcan /sbin/modprobe --ignore-install pcan
# PCAN - automatic made entry, end -----
```

ISA および PC / 104 PC CAN インターフェイスの標準割り当ては (io / irq) : 0x300 / 10、0x320 / 5 です。SP / EPP
モードでの PCAN-Dongle の標準割り当ては、(io / irq) : 0x378 / 7、0x278 / 5 です。


4 ドライバの使用法

インストール後、Udev システムがターゲットシステムで実行されている場合、ドライバは、PCI / PCIe ボードなどのように内部 PC CAN インターフェイスの次回の起動時、または、外部 PC CAN インターフェイス（USB アダプタなど）がシステムに接続されている場合にシステムによって自動的にロードされます。

4.1 ドライバの読み込み

ただし、モジュールであるため、システムを再起動せずに、PCAN モジュールをプローブするようにシステムに要求することでドライバをロードできます（root 権限が必要です）：

```
$ sudo modprobe pcan
```

 **注：** modprobe システムコマンドは、ドライバが依存する他のすべてのモジュールをロードするために管理します。代わりに insmod を使用する場合は、これらすべてのモジュールを手動でロードする必要があります：

```
$ modinfo pcan.ko | grep -e "^depends:"  
depends:          pcmcia,parport,i2c-algo-bit  
  
$ sudo modprobe pcmcia parport i2C-alog-bits  
$ sudo insmod pcan.ko
```

ドライバはカーネルに対してかなり冗長です：ドライバは、列挙する各 PC CAN インターフェイスのカーネルログバッファに 1 つまたは複数のメッセージを記録します。次に、何か問題が検出された場合にのみメッセージを保存します。

ロードされた直後にログに記録されるメッセージは次のとおりです。例を示します。

```
$ dmesg | grep pcan
[24612.510888] pcan: Release_YYYYMMDD_n (le)
[24612.510894] pcan: driver config [mod] [isa] [pci] [pec] [dng] [par] [usb] [pcc]
[24612.511057] pcan: uCAN PCI device sub-system ID 14h (4 channels)
[24612.511125] pcan 0000:01:00.0: irq 48 for MSI/MSI-X
[24612.511140] pcan: uCAN PCB v4h FPGA v1.0.5 (design 3)
[24612.511146] pcan: pci uCAN device minor 0 found
[24612.511148] pcan: pci uCAN device minor 1 found
[24612.511150] pcan: pci uCAN device minor 2 found
[24612.511153] pcan: pci uCAN device minor 3 found
[24612.516206] pcan: pci device minor 4 found
[24612.516230] pcan: pci device minor 5 found
[24612.516258] pcan: pci device minor 6 found
[24612.516280] pcan: pci device minor 7 found
[24612.516335] pcan: isa SJA1000 device minor 8 expected (io=0x0300,irq=10)
[24612.516369] pcan: isa SJA1000 device minor 9 expected (io=0x0320,irq=5)
[24612.516999] pcan: new high speed usb adapter with 2 CAN controller(s) detected
[24612.517237] pcan: PCAN-USB Pro FD (01h PCB01h) fw v2.1.0
[24612.517244] pcan: usb hardware revision = 1
[24612.517605] pcan: PCAN-USB Pro FD channel 1 device number=30
[24612.517729] pcan: usb device minor 0 found
[24612.517732] pcan: usb hardware revision = 1
[24612.518231] pcan: PCAN-USB Pro FD channel 2 device number=31
[24612.518354] pcan: usb device minor 1 found
[24612.522469] pcan: new usb adapter with 1 CAN controller(s) detected
[24612.522491] pcan: usb hardware revision = 28
[24612.579450] pcan: PCAN-USB channel device number=161
[24612.579453] pcan: usb device minor 2 found
[24612.579487] usbcore: registered new interface driver pcan
[24612.586265] pcan: major 249.
```

ドライバは、タイプに応じて各 PC CAN インターフェイスを列挙します。バージョン 8.5.1 までは、各タイプに次の範囲のデバイスマイナー番号がありました：

Hardware type	Minor number range
PCI/PCIe	[0 ... 7]
ISA and PC/104	[8 ... 15]
SP mode	[16 ... 23]
EPP mode	[24 ... 31]
USB	[32 ... 39]
PC-CARD	[40 ... 47]

表 3：デバイスのマイナー番号の範囲

v8.6.0 以降、CAN チャネルのニーズが高まっているため、ドライバは別のスキームに従って PC CAN インターフェイスを列挙します：

Hardware type	Minor number range
PCI/PCIe	[0 ... 31]
USB	[32 ... 63]
PC-CARD	[64 ... 71]
ISA and PC/104	[72 ... 79]
SP mode	[80 ... 87]
EPP mode	[88 ... 95]

表 4：デバイスのマイナー番号の範囲

この v8.6.0 の新しいスキームは、最初の USB デバイスチャンネル用に常にスロット 32 を予約しながら、ほとんどの使用済み PC CAN インターフェイスにより多くのスペースを提供します。

4.2 Udev ルール

Udev メカニズムは、システムが処理するデバイスの 1 つを認識するとき、起動時、またはハードウェアデバイスがシステムに接続されるときに、non-RT ドライバをロードします。

注：リアルタイムバージョンのドライバモジュールを実行すると、Udev システムに接続されていないリアルタイム（のみ）のデバイスが作成されるため、デバイスノードファイルは作成されません。

ドライバパッケージのインストールは、ドライバによって処理される CAN チャンネルを実装するデバイスノードをシステムが作成するのを支援するために、Udev にいくつかのデフォルトルールも追加します (peak-linuxdriver-xyz / driver / udev / 45-pcan.rules を参照)。デフォルトでは、Udev は CAN / CANFD チャンネルごとに / dev ディレクトリの下に 1 つの (character) デバイスノードを作成します。このデバイスノードの名前は次のとおりです。

- pcan プレフィックス
- PC CAN インターフェイスバスタイプ (pci、isa、usb...)
- CAN チャンネルが CAN-FD 対応の場合は fd サフィックス
- ユニークなマイナー番号

例えば：

```
$ ls -l /dev/pcan* | grep "^c"
crw-rw-rw- 1 root root 246, 8 févr. 3 14:59 /dev/pcanisa8
crw-rw-rw- 1 root root 246, 9 févr. 3 14:59 /dev/pcanisa9
crw-rw-rw- 1 root root 246, 4 févr. 3 14:59 /dev/pcanpci4
crw-rw-rw- 1 root root 246, 5 févr. 3 14:59 /dev/pcanpci5
crw-rw-rw- 1 root root 246, 0 févr. 3 14:59 /dev/pcanpcifd0
crw-rw-rw- 1 root root 246, 1 févr. 3 14:59 /dev/pcanpcifd1
crw-rw-rw- 1 root root 246, 2 févr. 3 14:59 /dev/pcanpcifd2
crw-rw-rw- 1 root root 246, 3 févr. 3 14:59 /dev/pcanpcifd3
crw-rw-rw- 1 root root 246, 35 févr. 3 15:24 /dev/pcanusb35
crw-rw-rw- 1 root root 246, 36 févr. 3 15:24 /dev/pcanusb36
crw-rw-rw- 1 root root 246, 32 févr. 3 14:59 /dev/pcanusbfd32
crw-rw-rw- 1 root root 246, 33 févr. 3 14:59 /dev/pcanusbfd33
crw-rw-rw- 1 root root 246, 34 févr. 3 14:59 /dev/pcanusbfd34
```

ドライバがインストールする Udev ルールにより、CAN チャンネルに関するより多くの情報を提供するいくつかのシンボリックリンクを作成できます：

1. Udev ルールは、CAN チャンネルごとに 1 つの / dev / pcanX を作成します
2. Udev ルールは、PC CAN インターフェイスに従って CAN チャンネルを、PC CAN インターフェイスの製品名で構成される同じサブディレクトリにグループ化します。
3. Udev のデフォルトルールは、CAN チャンネルが / sys の下に devid プロパティ (-1 とは異なる) をエクスポートする場合にも、他のシンボリックリンクを作成します (USB デバイスのように、ドライバの v8.10 以降の PCIe デバイスも同様です)。

以下の例は、`/dev/pcan*` ノード、シンボリックリンク、およびドライバで提供される Udev ルールが作成する可能性のあるサブディレクトリのリストを示しています。

```

$ ls -l /dev/pcan*
lrwxrwxrwx 1 root root      10 févr.  3 14:59 /dev/pcan0 -> pcanpcifd0
lrwxrwxrwx 1 root root      10 févr.  3 14:59 /dev/pcan1 -> pcanpcifd1
lrwxrwxrwx 1 root root      10 févr.  3 14:59 /dev/pcan2 -> pcanpcifd2
lrwxrwxrwx 1 root root      10 févr.  3 14:59 /dev/pcan3 -> pcanpcifd3
lrwxrwxrwx 1 root root      11 févr.  3 14:59 /dev/pcan32 -> pcanusbfd32
lrwxrwxrwx 1 root root      11 févr.  3 14:59 /dev/pcan33 -> pcanusbfd33
lrwxrwxrwx 1 root root      11 févr.  3 14:59 /dev/pcan34 -> pcanusbfd34
lrwxrwxrwx 1 root root       9 févr.  3 15:24 /dev/pcan35 -> pcanusb35
lrwxrwxrwx 1 root root       9 févr.  3 15:24 /dev/pcan36 -> pcanusb36
lrwxrwxrwx 1 root root       8 févr.  3 14:59 /dev/pcan4 -> pcanpci4
lrwxrwxrwx 1 root root       8 févr.  3 14:59 /dev/pcan5 -> pcanpci5
lrwxrwxrwx 1 root root       8 févr.  3 14:59 /dev/pcan8 -> pcanisa8
lrwxrwxrwx 1 root root       8 févr.  3 14:59 /dev/pcan9 -> pcanisa9
crw-rw-rw- 1 root root 246, 8 févr.  3 14:59 /dev/pcanisa8
crw-rw-rw- 1 root root 246, 9 févr.  3 14:59 /dev/pcanisa9
crw-rw-rw- 1 root root 246, 4 févr.  3 14:59 /dev/pcanpci4
crw-rw-rw- 1 root root 246, 5 févr.  3 14:59 /dev/pcanpci5
crw-rw-rw- 1 root root 246, 0 févr.  3 14:59 /dev/pcanpcifd0
crw-rw-rw- 1 root root 246, 1 févr.  3 14:59 /dev/pcanpcifd1
crw-rw-rw- 1 root root 246, 2 févr.  3 14:59 /dev/pcanpcifd2
crw-rw-rw- 1 root root 246, 3 févr.  3 14:59 /dev/pcanpcifd3
crw-rw-rw- 1 root root 246, 35 févr.  3 15:24 /dev/pcanusb35
crw-rw-rw- 1 root root 246, 36 févr.  3 15:24 /dev/pcanusb36
crw-rw-rw- 1 root root 246, 32 févr.  3 14:59 /dev/pcanusbfd32
crw-rw-rw- 1 root root 246, 33 févr.  3 14:59 /dev/pcanusbfd33
crw-rw-rw- 1 root root 246, 34 févr.  3 14:59 /dev/pcanusbfd34
lrwxrwxrwx 1 root root      11 févr.  3 14:59 /dev/pcanusbpfd32 -> pcanusbfd32
lrwxrwxrwx 1 root root      11 févr.  3 14:59 /dev/pcanusbpfd33 -> pcanusbfd33

/dev/pcan-pci:
total 0
drwxr-xr-x 2 root root 80 févr.  3 14:59 0
lrwxrwxrwx 1 root root 11 févr. 13 12:05 'devid=1' -> ../pcanpci4
lrwxrwxrwx 1 root root 11 févr. 13 12:05 'devid=2' -> ../pcanpci5

/dev/pcan-pcie_fd:
total 0
drwxr-xr-x 2 root root 80 févr.  3 14:59 0
drwxr-xr-x 2 root root 80 févr.  3 14:59 1
lrwxrwxrwx 1 root root 13 févr. 13 12:05 'devid=10' -> ../pcanpcifd0
lrwxrwxrwx 1 root root 13 févr. 13 12:05 'devid=11' -> ../pcanpcifd1
lrwxrwxrwx 1 root root 13 févr. 13 12:05 'devid=12' -> ../pcanpcifd2
lrwxrwxrwx 1 root root 13 févr. 13 12:05 'devid=13' -> ../pcanpcifd3

/dev/pcan-usb:
total 0
drwxr-xr-x 2 root root 60 févr.  3 15:24 0
drwxr-xr-x 2 root root 60 févr.  3 15:24 1
lrwxrwxrwx 1 root root 12 févr.  3 15:24 devid=161 -> ../pcanusb35

/dev/pcan-usb_fd:
total 0
drwxr-xr-x 2 root root 60 févr.  3 14:59 0
lrwxrwxrwx 1 root root 14 févr.  3 14:59 devid=12345678 -> ../pcanusbfd34

/dev/pcan-usb_pro_fd:
total 0
drwxr-xr-x 2 root root 80 févr.  3 14:59 0
lrwxrwxrwx 1 root root 14 févr.  3 14:59 devid=2 -> ../pcanusbfd32
lrwxrwxrwx 1 root root 14 févr.  3 14:59 devid=31 -> ../pcanusbfd33

```

これらの Udev ルールによって作成されたサブディレクトリの内容は、PC CAN インターフェイスごとに 1 つです。ツリー表現は、どの CAN チャンネルがどの PC CAN インターフェイスに接続されているかを示すためのより良い方法を提供します。

```
$ tree /dev/pcan-pci
1 directory, 4 files
1 directory, 4 files
```

```
$ tree /dev/pcan-pcie_fd
/dev/pcan-pcie_fd
├── 0
│   ├── can0 -> ../../pcanpcifd0
│   ├── can1 -> ../../pcanpcifd1
│   ├── can2 -> ../../pcanpcifd2
│   └── can3 -> ../../pcanpcifd3
├── 1
│   ├── can0 -> ../../pcanpcifd6
│   └── can1 -> ../../pcanpcifd7
├── devid=10 -> ../pcanpcifd0
├── devid=11 -> ../pcanpcifd1
├── devid=12 -> ../pcanpcifd2
└── devid=13 -> ../pcanpcifd3
2 directories, 10 files
```

```
$ tree /dev/pcan-usb
/dev/pcan-usb
├── 0
│   └── can0 -> ../../pcanusb35
├── 1
│   └── can0 -> ../../pcanusb36
└── devid=161 -> ../pcanusb35
2 directories, 3 files
```

```
$ tree /dev/pcan-usb_fd
/dev/pcan-usb_fd
├── 0
│   ├── can0 -> ../../pcanusbfd34
│   └── devid=12345678 -> ../pcanusbfd34
1 directory, 2 files
```

```
$ tree /dev/pcan-usb_pro_fd
/dev/pcan-usb_pro_fd
├── 0
│   ├── can0 -> ../../pcanusbfd32
│   └── can1 -> ../../pcanusbfd33
├── devid=2 -> ../pcanusbfd32
└── devid=31 -> ../pcanusbfd33
1 directory, 4 files
```

上記の構成では、ホストに接続された PCAN-USB Pro FD の 2 番目の CAN ポートを介して CAN バスにアクセスしたいユーザーアプリケーションは、無差別に開くことができます。

- / dev / pcanusbfd33
- / dev / pcan33
- / dev / pcan-usb_pro_fd / devid = 31
- / dev / pcan-usb_pro_fd / 0 / can1

i 注: 適切に構成され実行されている Udev システムを使用すると、これらのデバイスのファイルとディレクトリはすべてオンザフライで生成されます。ターゲットの non-RT システムに実行中の Udev システムがない場合は、ドライバのインストール後に毎回手でデバイスファイルを作成する必要があります。ドライバパッケージは、このためのシェルスクリプト `driver / pcan_make_devices` を提供します。たとえば、各タイプのデバイスを最大 2 つ作成するには、次のようにします。

```
$ cd driver
$ sudo ./pcan_make_devices 2
```

4.3 / proc インターフェイス

最初に行うテストの 1 つは、ドライバモジュールが正しくロードされ、期待どおりに実行されるかどうかを確認することです。そのためには、`/proc / pcan` 疑似ファイルを読み取ります。

```
$ cat /proc/pcan
*----- PEAK-System CAN interfaces (www.peak-system.com) -----
*----- Release_YYYYMMDD_n (X.Y.Z) MMM DD YYYY HH:MN:SS -----
*----- [mod] [isa] [pci] [pec] [dng] [par] [usb] [pcc] -----
*----- XX interfaces @ major 249 found -----
*n -type- -ndev- --base-- irq --btr- --read-- --write- --irqs-- -errors- status
0 pcifd -NA- f8c21000 048 0x001c 00000000 00000000 00000000 00000000 0x0000
1 pcifd -NA- f8c22000 048 0x001c 00000000 00000000 00000000 00000000 0x0000
2 pcifd -NA- f8c23000 048 0x001c 00000000 00000000 00000000 00000000 0x0000
3 pcifd -NA- f8c24000 048 0x001c 00000000 00000000 00000000 00000000 0x0000
4 pci -NA- fdee0000 016 0x001c 00000000 00000000 00000000 00000000 0x0000
5 pci -NA- fdee0400 016 0x001c 00000000 00000000 00000000 00000000 0x0000
6 pci -NA- fdee0800 016 0x001c 00000000 00000000 00000000 00000000 0x0000
7 pci -NA- fdee0c00 016 0x001c 00000000 00000000 00000000 00000000 0x0000
8 isa -NA- 300 010 0x001c 00000000 00000000 00000000 00000000 0x0000
9 isa -NA- 320 005 0x001c 00000000 00000000 00000000 00000000 0x0000
32 usbfd -NA- 3 030 0x001c 00000000 00000000 00000000 00000000 0x0000
33 usbfd -NA- 3 031 0x001c 00000000 00000000 00000000 00000000 0x0000
34 usb -NA- ffffffff 161 0x001c 00000000 00000000 00000000 00000000 0x0000
```

`/proc / pcan` ファイルには次のものが含まれています：

- ビルド日時を含むドライババージョン（リリース日とバージョン番号）
- ドライバが処理できる PC CAN インターフェイスのリスト（9 ページの表 1 を参照）
- ドライバによって検出された PC CAN インターフェイスの数と、Linux カーネルがドライバに割り当てたメジャー番号
- ドライバが検出したすべての CAN デバイスのテーブル（1 行に 1 つ）

PC CAN インターフェイステーブルの列は、各インターフェイスに共通のプロパティです：

Column	PC CAN interface property description	
n	decimal value	ドライバが PC CAN インターフェイスに割当てたマイナー番号
type	pci	FPGA SJA1000 を搭載した PCI / PCIe / PCC / EC ベースのインターフェイス
	isa	SJA1000 コントローラを搭載した ISA ベースのインターフェイス
	sp	SJA1000 コントローラを搭載した Standard (標準) パラレルインターフェイス
	epp	SJA1000 コントローラを搭載した Enhanced (拡張) パラレルインターフェイス
	usb	SJA1000 コントローラを搭載した USB インターフェイス (PCAN-USB)
	usbfd	CAN FD FPGA を搭載した USB インターフェイス (PCAN-USB FD)
	pcifd	CAN FD FPGA を搭載した PCI / PCIe ベースのインターフェイス
ndev	canx	ドライバのビルド時に netdev インターフェイスが選択されている場合、この列には、SocketCAN レイヤの PC CAN インターフェイスの名前が含まれます。
	not applicable	ドライバが chardev モード (デフォルトモード) で実行するように構築されている場合、この列は無意味です
base	hexadecimal value	パラレルまたは ISA インターフェイスの場合、PC CAN インターフェイスハードウェアへのアクセスに使用される I / O ポート
		その他の場合に PC CAN インターフェイスハードウェアにアクセスするための I / O ベースアドレス
		PC CAN インターフェイスが USB インターフェイスの場合、アダプタのシリアル番号
irq	decimal value	PC CAN インターフェイスに添付されている IRQ 番号 (ある場合)
		PC CAN インターフェイスが USB インターフェイスの場合、PC CAN インターフェイスに設定されたデバイス番号
btr	hexadecimal value	SJA1000 ビットレートレジスタの BTR0BTR1 フォーマットに従って、PC CAN インターフェイスに設定された nominal ビットレート
read	hexadecimal value	このインターフェイスを使用したアプリケーションによってドライバから読み取られた CAN / CAN FD フレームの数
write	hexadecimal value	このインターフェイスを使用したアプリケーションによってドライバに書き込まれた CAN / CAN FD フレームの数
irqs	hexadecimal value	その PCCAN インターフェイスのドライバによってカウントされた割り込みの数 (ドライバがハンドラを IRQ レベルに接続した場合)
		USB CAN インターフェイスの場合、ドライバが USB サブシステムから受信したパケットの数
errors	hexadecimal value	このインターフェイスのドライバで発生したエラーの数。このカウンターは、あらゆる種類のエラー (コントローラエラーおよびドライバ内部エラー) を処理します。エラーに関する詳細は、ステータス列に記載されています。
status	bit mask	各エラービットの意味は、/usr/include/pcan.h で定義されている CAN_ERR_xxx 定数によって定義されます。

表 5: /proc/pcan コラム

4.4 / sysfs インターフェイス



注：この機能は、ドライバの v8.x 以降の新機能です。

歴史的な理由から、ドライバの v8.x は常に / proc / pcan ファイルを処理しますが、これは非推奨であり、CAN 2.0 でのみ使用するためのものと見なす必要があります。v8.x 以降、ドライバはすべての / proc / pcan プロパティ（およびその他のプロパティ）も / sysfs インターフェイスにエクスポートします。

- a) / sys / class / pcan / version 属性は、ドライバのバージョン番号をエクスポートします：

```
$ cat /sys/class/pcan/version
8.0.0
```

- b) v8.11.0 以降、/ sys / class / pcan / clk_ref 属性は、ドライバが使用するクロックリファレンスをエクスポートします：

```
$ cat /sys/class/pcan/clk_ref
0
```

この数値は、/ usr / include / linux / time.h で定義されているいくつかの CLOCK_XXX 値に対応しています：

Variable	Mnemonic	Description
0	CLOCK_REALTIME	このクロックは、システム時間の不連続なジャンプ（たとえば、システム管理者が手動でクロックを変更した場合）、および adjtime (3) と NTP によって実行される増分調整の影響を受けます。
1	CLOCK_MONOTONIC	このクロックは、システム時間の不連続なジャンプの影響を受けませんが（たとえば、システム管理者が手動でクロックを変更した場合）、 adjtime (3) および NTP によって実行される増分調整の影響を受けます。
4	CLOCK_MONOTONIC_RAW	CLOCK_MONOTONIC に似ていますが、NTP 調整または adjtime (3) によって実行される増分調整の対象とならない生のハードウェアベースの時間へのアクセスを提供します。
7	CLOCK_BOOTTIME	システムが一時停止されている時間を含むことを除いて、 CLOCK_MONOTONIC と同じです。これにより、アプリケーションは、 settimeofday (2) を使用して時刻を変更すると不連続になる可能性がある、 CLOCK_REALTIME の複雑さに対処することなく、サスペンド対応の単調クロックを取得できます。

表 6：ドライバがタイムスタンプに使用するクロック参照

- c) /sys/class/pcan ディレクトリは、処理するすべての CAN インターフェイスのリストをエクスポートします:


```
$ tree -a /sys/class/pcan
/sys/class/pcan
├── pcanisa8 -> ../../devices/virtual/pcan/pcanisa8
├── pcanisa9 -> ../../devices/virtual/pcan/pcanisa9
├── pcanpci4 -> ../../devices/virtual/pcan/pcanpci4
├── pcanpci5 -> ../../devices/virtual/pcan/pcanpci5
├── pcanpcifd0 -> ../../devices/virtual/pcan/pcanpcifd0
├── pcanpcifd1 -> ../../devices/virtual/pcan/pcanpcifd1
├── pcanpcifd2 -> ../../devices/virtual/pcan/pcanpcifd2
├── pcanpcifd3 -> ../../devices/virtual/pcan/pcanpcifd3
├── pcanusb35 -> ../../devices/virtual/pcan/pcanusb35
├── pcanusb36 -> ../../devices/virtual/pcan/pcanusb36
├── pcanusbfd32 -> ../../devices/virtual/pcan/pcanusbfd32
├── pcanusbfd33 -> ../../devices/virtual/pcan/pcanusbfd33
├── pcanusbfd34 -> ../../devices/virtual/pcan/pcanusbfd34
└── version
```

- d) これらのエントリは、以下の例 (太字) に示すように、一部の PCAN デバイスのプライベートプロパティをエクスポートするように拡張されています (太字-緑色の線のプロパティは /proc/pcan の列と同じです):

```
$ ls -l /sys/class/pcan/pcanpci4/
total 0
-r--r--r-- 1 root root 4096 nov. 6 12:34 adapter_name
-r--r--r-- 1 root root 4096 nov. 6 12:34 adapter_number
-r--r--r-- 1 root root 4096 nov. 6 12:34 adapter_version
-r--r--r-- 1 root root 4096 nov. 6 12:34 base
-r--r--r-- 1 root root 4096 nov. 6 12:34 btr0btrl
-r--r--r-- 1 root root 4096 nov. 6 12:34 bus_state
-r--r--r-- 1 root root 4096 nov. 6 12:34 clk_drift
-r--r--r-- 1 root root 4096 nov. 6 12:34 clock
-r--r--r-- 1 root root 4096 nov. 6 12:34 ctrlr_number
-r--r--r-- 1 root root 4096 nov. 6 12:34 dev
-rw-r--r-- 1 root root 4096 nov. 6 12:34 devid
-r--r--r-- 1 root root 4096 nov. 6 12:34 errors
-r--r--r-- 1 root root 4096 nov. 6 12:34 hwtype
-r--r--r-- 1 root root 4096 nov. 6 12:34 init_flags
-r--r--r-- 1 root root 4096 nov. 6 12:34 irq
-r--r--r-- 1 root root 4096 nov. 6 12:34 irqs
-r--r--r-- 1 root root 4096 nov. 6 12:34 minor
-r--r--r-- 1 root root 4096 nov. 6 12:34 ndev
-r--r--r-- 1 root root 4096 nov. 6 12:34 nom_bitrate
-r--r--r-- 1 root root 4096 nov. 6 12:34 nom_brp
-r--r--r-- 1 root root 4096 nov. 6 12:34 nom_sjw
-r--r--r-- 1 root root 4096 nov. 6 12:34 nom_tq
-r--r--r-- 1 root root 4096 nov. 6 12:34 nom_tseg1
-r--r--r-- 1 root root 4096 nov. 6 12:34 nom_tseg2
drwxr-xr-x 2 root root 0 nov. 6 12:34 power
-r--r--r-- 1 root root 4096 nov. 6 12:34 read
-r--r--r-- 1 root root 4096 nov. 6 12:34 rx_fifo_ratio
-r--r--r-- 1 root root 4096 nov. 6 12:34 serialno
-r--r--r-- 1 root root 4096 nov. 6 12:34 status
lrwxrwxrwx 1 root root 0 nov. 6 12:34 subsystem -> ../../../../../../class/pcan
-r--r--r-- 1 root root 4096 nov. 6 12:34 tx_fifo_ratio
-r--r--r-- 1 root root 4096 nov. 6 12:34 type
-rw-r--r-- 1 root root 4096 nov. 6 12:33 uevent
-r--r--r-- 1 root root 4096 nov. 6 12:34 write
```

e) 前記のすべてのファイルの内容を読取ると、次のように表示されます。

```
$ for f in /sys/class/pcan/pcanpci4/*; do [ -f $f ] && echo -n "`basename $f` =
" && cat $f; done
adapter_name = PCAN-PCI
adapter_number = 0
adapter_version =
base = 0xfdee0000
btr0btr1 = 0x001c
bus_state = 0
clk_drift = 0
clock = 8000000
ctrlr_number = 0
dev = 249:4
devid = 4294967295
errors = 0
hwtype = 10
init_flags = 0x00000000
irq = 16
irqs = 0
minor = 4
ndev = can4
nom_bitrate = 500000
nom_brp = 1
nom_sjw = 1
nom_tq = 125
nom_tseg1 = 13
nom_tseg2 = 2
read = 0
rx_fifo_ratio = 0.00
serialno = 4294967295
status = 0x0000
tx_fifo_ratio = 0.00
type = pci
uevent = MAJOR=249
MINOR=4
DEVNAME=pcanpci4
write = 0
```

 **注:** CAN ハードウェアやドライバがコンパイルされたモードによっては、デバイスノードがさらにいくつかのプロパティをエクスポートする場合があります。たとえば、CAN FD PCIe デバイスは次のプロパティをエクスポートします (新しいプロパティは**太字**で示されています)。

```
$ for f in /sys/class/pcan/pcanpcifd1/*; do [ -f $f ] && echo -n "`basename $f`
" && cat $f; done
adapter_name = PCAN-PCIe FD
adapter_number = 0
adapter_version = 2.1.3
base = 0xf8ba1000
btr0btr1 = 0x001c
bus_load = 0
bus_state = 0
clock = 80000000
ctrlr_number = 1
data_bitrate = 2000000
data_brp = 2
data_sample_point = 7500
data_sjw = 1
data_tq = 25
```

```

data_tseg1 = 14
data_tseg2 = 5
dev = 249:1
devid = 4294967295
errors = 0
hwtype = 19
irq = 48
irqs = 0
minor = 0
nom_bitrate = 500000
nom_brp = 4
nom_sample_point = 8750
nom_sjw = 1
nom_tq = 50
nom_tseg1 = 34
nom_tseg2 = 5
read = 0
rx_error_counter = 0
rx_fifo_ratio = 0.00
status = 0x0000
tx_error_counter = 0
tx_fifo_ratio = 0.00
type = pcifd
uevent = MAJOR=249
MINOR=1
DEVNAME=pcanpcifd1
write = 0

```

- f) 一部のエントリは書き込み許可で公開されています。これらのエントリは書き込むことができますが、root 権限が必要です：

たとえば、自分のデバイス番号を CAN チャンネルに添付することは、sysfs を介して（も）可能です：

```

$ cat /sys/class/pcan/pcanusb32/devid
4294967295
$ echo 12 | sudo tee /sys/class/pcan/pcanusb32/devid
[sudo] password for user:
12
$ cat /sys/class/pcan/pcanusb32/devid
12

```

注：v8.10 以降、ms を書き込むことで sysfs を介して CAN チャンネルを識別することもできます：“led” プロパティに遅延すると、この遅延中にチャンネル LED が点滅します。たとえば、“/dev/pcanusb32” の LED を 3 秒間切り替えるには、次のようにします：

```
$ echo 3000 | sudo tee /sys/class/pcan/pcanusb32/led
```

4.5 lspcan ツール

注：この機能は、ドライバの v8.x 以降の新機能です。

lspcan ツールは、/ sysfs インターフェイスに基づくシェルスクリプトであり、ホストの PC CAN インターフェイスと CAN チャンルの情報を取得するために使用できます。

```
$ ./lspcan --help
lspcan: ドライバによって検出されたPEAK-System CAN / CAN FDデバイスのリスト

Option:
-a | --all          すべて同等: -i -s
-f | --forever     デバイスで永久ループ (^Cで停止)
-h | --help        このヘルプを表示する
-i | --info        PCANデバイスに関する情報
-s | --stats       PCANデバイスに関する統計
-t | --title       列の上にタイトル行を表示
-T | --tree        ツリーバージョン
--version         ドライババージョンの表示
```

“-i” オプションは、デバイスノードのプロパティを表示します：

```
$ ./lspcan -T -t -i
dev name          port  irq   clock  btrs   bus
[PCAN-ISA 0]
|_ pcanisa8       CAN1  10    8MHz   500k   CLOSED
|_ pcanisa9       CAN2  5     8MHz   500k   CLOSED
[PCAN-PCI 0]
|_ pcanpci4       CAN1  19    8MHz   500k   CLOSED
|_ pcanpci5       CAN2  19    8MHz   500k   CLOSED
[PCAN-PCIe FD 0]
|_ pcanpcifd0     CAN1  32    80MHz  500k+2M  CLOSED
|_ pcanpcifd1     CAN2  32    80MHz  500k+2M  CLOSED
[PCAN-PCIe FD 1]
|_ pcanpcifd2     CAN1  33    80MHz  500k+2M  CLOSED
|_ pcanpcifd3     CAN2  33    80MHz  500k+2M  CLOSED
[PCAN-USB 0]
|_ pcanusb32      CAN1  -     8MHz   500k   CLOSED
[PCAN-USB 1]
|_ pcanusb33      CAN1  -     8MHz   500k   CLOSED
[PCAN-USB Pro FD 0]
|_ pcanusbfd34    CAN1  -     80MHz  500k+2M  CLOSED
|_ pcanusbfd35    CAN2  -     80MHz  500k+2M  CLOSED
```

一方、-T -t -s -f を指定して lspcan を実行すると、Linux ホストに存在するすべての PC CAN インターフェイスから収集された統計の詳細ビューで画面が每秒更新されます。

```

PCAN driver version: 8.x.y
dev name          port  irq  clock  btrs   bus    %bus   rx    tx
err
[PCAN-ISA 0]
|_ pcanisa8      CAN1  10   8MHz   500k   CLOSED -     0     0     0
|_ pcanisa9      CAN2   5   8MHz   500k   CLOSED -     0     0     0
[PCAN-PCI 0]
|_ pcanpci4      CAN1  19   8MHz   500k   CLOSED -     0     0     0
|_ pcanpci5      CAN2  19   8MHz   500k   CLOSED -     0     0     0
[PCAN-PCIe FD 0]
|_ pcanpcifd0    CAN1  30   80MHz  500k+2M CLOSED 0.00  0     0     0
|_ pcanpcifd1    CAN2  30   80MHz  500k+2M CLOSED 0.00  0     0     0
[PCAN-PCIe FD 1]
|_ pcanpcifd2    CAN1  31   80MHz  500k+2M CLOSED 0.00  0     0     0
|_ pcanpcifd3    CAN2  31   80MHz  500k+2M CLOSED 0.00  0     0     0
[PCAN-USB 0]
|_ pcanusb35     CAN1  -    8MHz   500k   CLOSED -     0     0     0
[PCAN-USB 1]
|_ pcanusb36     CAN1  -    8MHz   1M     PASSIVE -    535608 0
585
[PCAN-USB Pro FD 0]
|_ pcanusbfd32   CAN1  -    80MHz  500k+2M CLOSED 0.00  0     0     0
|_ pcanusbfd33   CAN2  -    80MHz  1M     ACTIVE 10.01  1    535634 0
[PCAN-USB FD 0]
|_ pcanusbfd34   CAN1  -    80MHz  500k+2M CLOSED 0.00  0     0     0

```



注：上記の画面コピーの内容は、ドライバのバージョンによって変更される場合があります。

4.6 read / write インターフェイス

説明したように、`/proc/pcan` を読み取ると、ロードされると、ドライバは検出した CAN チャンネルで動作する準備が整います。それぞれについて、チャンネルからの `read`、チャンネルへの `write` を可能にするデフォルトのビットレート構成が定義されています。`chardev` モードでは、ドライバの `chardev` インターフェイスの `read / write` エントリは次の通りです。

- CAN チャンネルを初期化
- CAN / CAN FD フレームの `write`
- CAN / CAN FD フレームの `read`

このシンプルなインターフェイスにより、ドライバが正しく機能するかどうかを確認できます。このインターフェイスは、以下で構成される構文を使用します。

1. コマンドを示す `character`
2. コマンドのパラメータのリスト

コマンドとパラメータは空白文字で区切る必要があります。

Command	Parameter	Description																		
i	XXXX	XXXX が数値 ≤ 65535 の場合、BTR0BTR1 SJA1000 レジスタ値として解釈されます。その後、CAN チャンネルは CAN 2.0 モードでのみ対応するビットレート値で初期化されます。																		
	param1=value1[,param2=value2...]	<p>パラメータが数値でない場合、param=value の組み合わせのリストで構成される文字列として解析されます。各カプトルは次のカプトルと"," (カンマ) で区切られます。パラメータ リストは次のとおりです。</p> <table border="1"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>f_clock</td> <td>選択するクロック。</td> </tr> <tr> <td>nom_bitrate</td> <td>nominal ビットレート (ビット/秒)。</td> </tr> <tr> <td>nom_brp</td> <td rowspan="4">ISO 11898 で定義されているnominal ビットレートのビット タイミング仕様。</td> </tr> <tr> <td>nom_tseg1</td> </tr> <tr> <td>nom_tseg2</td> </tr> <tr> <td>nom_sjw</td> </tr> <tr> <td>data_bitrate</td> <td>CAN チャンネルが CAN FD モードで初期化される場合のデータ ビットレート (ビット/秒)。</td> </tr> <tr> <td>data_brp</td> <td rowspan="4">チャンネルが CAN FD モードで初期化される場合の、ISO 11898 で定義されたデータ ビットレートのビット タイミング仕様。</td> </tr> <tr> <td>data_tseg1</td> </tr> <tr> <td>data_tseg2</td> </tr> <tr> <td>data_sjw</td> </tr> </tbody> </table> <p>各値は数値です。k や M などの単位記号をショートカットとして使用できます。</p> <p>例： \$ echo "i nom_bitrate=1M" > /dev/pcanusb0</p> <p>上記のコマンドは、pcanusb0 CAN チャンネルを初期化して、1 Mbit/s CAN 2.0 チャンネルに接続します。</p>	Parameter	Description	f_clock	選択するクロック。	nom_bitrate	nominal ビットレート (ビット/秒)。	nom_brp	ISO 11898 で定義されているnominal ビットレートのビット タイミング仕様。	nom_tseg1	nom_tseg2	nom_sjw	data_bitrate	CAN チャンネルが CAN FD モードで初期化される場合のデータ ビットレート (ビット/秒)。	data_brp	チャンネルが CAN FD モードで初期化される場合の、ISO 11898 で定義されたデータ ビットレートのビット タイミング仕様。	data_tseg1	data_tseg2	data_sjw
Parameter	Description																			
f_clock	選択するクロック。																			
nom_bitrate	nominal ビットレート (ビット/秒)。																			
nom_brp	ISO 11898 で定義されているnominal ビットレートのビット タイミング仕様。																			
nom_tseg1																				
nom_tseg2																				
nom_sjw																				
data_bitrate	CAN チャンネルが CAN FD モードで初期化される場合のデータ ビットレート (ビット/秒)。																			
data_brp	チャンネルが CAN FD モードで初期化される場合の、ISO 11898 で定義されたデータ ビットレートのビット タイミング仕様。																			
data_tseg1																				
data_tseg2																				
data_sjw																				
m	s id len [xx [xx ...]]	<p>xx [xx] で指定された len データ バイトで CAN 標準メッセージ ID (数値 $\leq 0x7ff$) を書き込みます。</p> <p>例： \$ echo "m s 0x123 3 01 02 03" > /dev/pcanusb0</p> <p>上記のコマンドは、USB CAN インターフェイスの 1 番目の CAN ポートに接続された CAN バスに、データ バイト「01 02 03」を 3 つ含む CAN メッセージ ID 0x123 を書き込みます。</p>																		
	e id len [xx [xx ...]]	<p>xx [xx] で指定された len データ バイトで CAN 拡張メッセージ ID (数値 $\leq 0x3ffffff$) を書き込みます。</p> <p>例： \$ echo "m e 0x123 3 01 02 03" > /dev/pcanusb0</p> <p>上記のコマンドは、USB CAN インターフェイスの 1 番目の CAN ポートに接続された CAN バスに、データ バイト「01 02 03」を 3 つ含む CAN メッセージ ID 0x00000123 を書き込みます。</p>																		
r	s id	標準 id (数値 $\leq 0x7ff$)のCAN RTR(Remote Transmission Request)を書き込みます。																		
	e id	拡張 id (数値 $\leq 0x7ff$)のCAN RTR(Remote Transmission Request)を書き込みます。																		
M	m と同じですが、ドライバに自己受信機能をアクティブにするように要求します (指定されたチャンネルの CAN コントローラが発信 CAN フレームを独自の rx キューにコピーできる場合)。																			
R	r と同じですが、ドライバに自己受信機能をアクティブにするように要求します (指定されたチャンネルの CAN コントローラが発信 CAN フレームを独自の rx キューにコピーできる場合)。																			
b	m と同じですが、ドライバに BRS 機能をアクティブにするように求めます (指定されたチャンネルに CAN FD コントローラが装備されている場合)。																			
B	b と同じですが、ドライバに自己受信機能をアクティブにするように要求します (指定されたチャンネルの CAN FD コントローラが発信 CAN FD フレームを独自の rx キューにコピーできる場合)。																			

表 7: read / write インターフェイスのシンタックス

インターフェイスから読み取る場合、ユーザーは上記のメッセージのいずれかとステータス (x) メッセージを受信できます。

Message	Parameter	Description										
x	b id len [xx [xx ...]]	CANバスの状態を示すバスステータスメッセージ :										
		<table border="1"><thead><tr><th>id</th><th>Bus State</th></tr></thead><tbody><tr><td>1</td><td>ACTIVE</td></tr><tr><td>2</td><td>WARNING</td></tr><tr><td>3</td><td>PASSIVE (パッシブ)</td></tr><tr><td>4</td><td>BUSOFF (バスオフ)</td></tr></tbody></table>	id	Bus State	1	ACTIVE	2	WARNING	3	PASSIVE (パッシブ)	4	BUSOFF (バスオフ)
		id	Bus State									
		1	ACTIVE									
		2	WARNING									
	3	PASSIVE (パッシブ)										
	4	BUSOFF (バスオフ)										
	c id len [xx [xx ...]]	コントローラのエラー (error) /ステータス (status) :										
		<table border="1"><thead><tr><th>id</th><th>Error</th></tr></thead><tbody><tr><td>5</td><td>コントローラの Rx キューがエンプティ</td></tr><tr><td>6</td><td>コントローラの Rx キューがオーバーフロー</td></tr><tr><td>7</td><td>コントローラの Tx キューがエンプティ</td></tr><tr><td>8</td><td>コントローラの Tx キューがオーバーフロー</td></tr></tbody></table>	id	Error	5	コントローラの Rx キューがエンプティ	6	コントローラの Rx キューがオーバーフロー	7	コントローラの Tx キューがエンプティ	8	コントローラの Tx キューがオーバーフロー
		id	Error									
		5	コントローラの Rx キューがエンプティ									
	6	コントローラの Rx キューがオーバーフロー										
	7	コントローラの Tx キューがエンプティ										
8	コントローラの Tx キューがオーバーフロー											
i id len [xx [xx ...]]	内部 (ドライバ) のエラー (error) /ステータス (status) :											
	<table border="1"><thead><tr><th>id</th><th>Error</th></tr></thead><tbody><tr><td>5</td><td>ドライバの Rx キューがエンプティ</td></tr><tr><td>6</td><td>ドライバの Rx キューがオーバーフロー</td></tr><tr><td>7</td><td>ドライバの Tx キューがエンプティ</td></tr><tr><td>8</td><td>ドライバの Tx キューがオーバーフロー</td></tr></tbody></table>	id	Error	5	ドライバの Rx キューがエンプティ	6	ドライバの Rx キューがオーバーフロー	7	ドライバの Tx キューがエンプティ	8	ドライバの Tx キューがオーバーフロー	
	id	Error										
	5	ドライバの Rx キューがエンプティ										
6	ドライバの Rx キューがオーバーフロー											
7	ドライバの Tx キューがエンプティ											
8	ドライバの Tx キューがオーバーフロー											

表 8 : ステータス (x) メッセージ

4.7 test ディレクトリ

`peak-linux-driver-X.Y.Z.tar.gz` には、C/C++ソースと Makefile を含む test ディレクトリが含まれており、`chardev` インストール全体 (ドライバとライブラリ) が完全に機能しているかどうかを確認するために、簡単なテストバイナリアプリケーションをすばやくビルドして実行できます。これらのテストプログラムは、RT 環境だけでなく non-RT 環境でのドライバライブラリの使用法を示すサンプルプログラムでもあります。

test ディレクトリアプリケーションは、lib ディレクトリの下でライブラリがビルドおよびインストールされた後にビルドする必要があります。ドライバと同様に、これらのライブラリとアプリケーションは non-RT および RT コンパイルを受け入れます。

7 ページの「3.1 バイナリのビルド」で説明されているグローバルパッケージのインストールにより、これらのバイナリがシステムにビルドおよびインストールされました。それらを (再) ビルドするには (RT システムコールを使用せずに) :

```
$ cd peak-linux-driver-x.y.z
$ make -C test
```

32-bit version :

ドライババージョン 8.3 以降、64 ビットバージョンの pcan ドライバは任意の 32 ビットアプリケーションで動作します。このテストディレクトリに保存されているアプリケーションの 32 ビットバージョンをビルドするには、次の手順を実行する必要があります。

```
$ cd peak-linux-driver-x.y.z
$ make -C test a1132
```

注 : 最初に 32 ビットバージョンの `libpcan` をビルドしてインストールしておく必要があります (32 ビットバージョンのライブラリの作成ページ 8 を参照)。さらに、64 ビットカーネルの実行中に 32 ビットアプリケーションをビルドするには、最初に `gcc-multilib` パッケージをインストールする必要があります。最後に、特定の `libpopt32` ビットパッケージを次の場所にインストールする必要があります :

```
$ sudo apt-get install gcc-multilib
$ sudo apt-get install libpopt-dev:i386
```

▶ リアルタイムバージョン :


バイナリの RT バージョンを再構築したいユーザーは、次のことを行う必要があります。

```
$ cd peak-linux-driver-x.y.z
$ make -C test RT=XENOMAI # Or "make xeno" since pcan 8.2
```

Xenomai RT 拡張カーネルを実行している場合、

```
$ cd peak-linux-driver-x.y.z
$ make -C test RT=RTAI # Or "make rtai" since pcan 8.2
```

RTAI 拡張カーネルを実行している場合。

 **注 :** CAN-FD 固有のアプリケーションのユーザー（開発者）は、33 ページの 4.7.5 で説明されている新しい pcanfdtst アプリケーションを直接見ることができます。

4.7.1 receivetest

このアプリケーションは、特定の CAN 2.0 チャネル（のみ！）から受信したすべてのフレームを stdout に書き込みます。このアプリケーションは、RT 環境と non-RT 環境の両方での古い libpcan CAN 2.0 API の使用法も示しています。

使用法 :

```
$ receivetest --help

receivetest Version "Release_20150611_n" (www.peak-system.com)
----- Copyright (C) 2004-2009 PEAK System-Technik GmbH -----
receivetest comes with ABSOLUTELY NO WARRANTY. This is free software and you are welcome
to redistribute it under certain
conditions. For details see attached COPYING file.

receivetest - a small test program which receives and prints CAN messages.
usage: receivetest [-b=BTR0BTR1] [-e] [-?]
                {[-f=devicenode] | {[-t=type] [-p=port [-i=irq]]}}

options:
-f=devicenode  path to PCAN device node (default=/dev/pcan0)
-t=type        type of interface (pci, sp, epp, isa, pccard, usb (default=pci)
-p=port        port number if applicable (default=1st port of type)
-i=irq         irq number if applicable (default=irq of 1st port)
-b=BTR0BTR1   bitrate code in hex (default=see /proc/pcan)
-e            accept extended frames (default=standard frames only)
-d=no         donot display received messages (default=yes)
-n=mloop      number of loops to run before exit (default=infinite)
-? or -help   displays this help

receivetest: finished (0): 0 message(s) received
```

例：

CAN バスに接続された USB インターフェイスの 1 番目の CAN ポートから受信した最大 100 の (extended : 拡張 および standard : 標準) メッセージを 1 Mbit/s で表示します。

```
$ receivetest -f=/dev/pcanusb32 -b=0x14 -e -n=100
```

i **注:** このプログラムによって CAN インターフェイスに設定されたビットレートは、ドライバによってエクスポートされます。

```
$ cat /proc/pcan | grep -e "^32"
32 usb      -NA-          3 030 0x0014 00000001 00000000 00000000 00000001 0x0000
$ cat /sys/class/pcan/pcanusb32/nom_bitrate
1000000
$ cat /sys/class/pcan/pcanusb32/btr0btr1
0x0014
```

i **注:** RT Linux の実行中、RT デバイスは "/dev" の下に表示されないため、CAN_Open (libpcan) の RT バージョンでは、デバイス名の文字列から "/dev" プレフィックスが削除されますが、pcanfd_open (lipcanfd) では削除されません。この回避策は、"/dev/pcanX" デバイス名でのみ機能します。

4.7.2 transmitest

このアプリケーションは、特定のテキストファイルで見つかったすべてのフレームを特定の CAN 2.0 チャンネルにライントします (のみ!)。このアプリケーションは、RT 環境と非 RT 環境の両方で古い lipcan CAN 2.0 API を使用する方法も示しています。

使用法：

```
$ transmitest --help

transmitest Version "Release_20150610_n" (www.peak-system.com)
----- Copyright (C) 2004-2009 PEAK System-Technik GmbH -----
transmitest comes with ABSOLUTELY NO WARRANTY. This is free software and you are
welcome to redistribute it under certain conditions. For details see attached
COPYING file.

transmitest - a small test program which transmits CAN messages.
usage: transmitest filename
           [-b=BTR0BTR1] [-e] [-r=msec] [-n=max] [-?]
           {[-f=devicenode] | {[-t=type] [-p=port [-i=irq]]}}
Filename  mandatory name of message description file.
options:
-f=devicenode path to PCAN device node (default=/dev/pcan0)
-t=type       type of interface (pci, sp, epp, isa, pccard, usb (default=pci))
-p=port       port number if applicable (default=1st port of type)
-i=irq        irq number if applicable (default=irq of 1st port)
-b=BTR0BTR1  bitrate code in hex (default=see /proc/pcan)
-e           accept extended frames (default=standard frames only)
-r=msec       max time to sleep before transm. next msg (default=no sleep)
-n=loop       number of loops to run before exit (default=infinite)
-? or -help  displays this help

transmitest: finished (0).
```

ファイル transmit.txt は、test ディレクトリの例として示されています。このファイルの構文は非常に単純で、ドライバのライトインターフェイスの構文に従います。テストは、入力テキストファイルで見つかったフレームの送信をループします。コマンドラインで -n オプションを指定しない限り、ループの数は無限です。

例：

transmit.txt に記載されているすべての CAN2.0 フレームを 1 Mbit / s で CAN バスに接続された USB インターフェイスの最初の CAN ポートへ 100 回送信します：

```
$ transmitest transmit.txt -f=/dev/pcanusb32 -b=0x14 -e -n=100
```

i **注：**このプログラムによってこの CAN インターフェイスに設定されたビットレートは、ドライバによってエクスポートされます。

```
$ cat /proc/pcan | grep -e "^32"
32 usb      -NA-          3 030 0x0014 00000001 00000000 00000000 00000001 0x0000
$ cat /sys/class/pcan/pcanusb32/nom_bitrate
1000000
$ cat /sys/class/pcan/pcanusb32/btr0btr1
0x0014
```

i **注：**RT Linux の実行中、RT デバイスは “/dev” の下に表示されないため、CAN_Open (libpcan) の RT バージョンでは、デバイス名の文字列から “/dev” プレフィックスが削除されますが、pcanfd_open (lipcanfd) では削除されません。この回避策は、“/dev/pcanX” デバイス名でのみ機能します。

4.7.3 pcan-settings

このアプリケーションを使用すると、一部の PC CAN インターフェイスの不揮発性メモリとの間で特定の値を読み書きできます。この機能は、hot-pluggable CAN インターフェイスに、接続されているソケットに関係なく、常に同じデバイスノード名を持たせたいユーザーに役立ちます（オペレーティングシステムのデバイス列挙ルールでは、同じデバイスに同じ番号が与えられません。デバイスが同じ socket / bus / port に接続されていません...）。

ドライババージョン 8.8 以降、pcan-settings を使用すると、スーパーユーザーは特定の PC CAN インターフェイスを “USB Mass Storage Device” モードに切り替えることができます。このモードは、PC CAN インターフェイスを新しいファームウェアでアップグレードするために使用されます（42 ページの 4.9 USB 大容量ストレージデバイスモードも参照）。

使用法：

```
$ pcan-settings --help
Usage: pcan-settings [OPTION...]
  -f, --deviceNode='device file path'  Set path to PCAN device (default:
                                         "/dev/pcan32")
  -s, --SerialNo                        Get serial number
  -d, --DeviceNo[='device number']     Get or set device number
  -v, --verbose                          Make it verbose
  -q, --quiet                            No display at all
  -M, --MSD                             Switch device in Mass Storage Device
                                         mode (root privileges needed)

Help options:
  -?, --help                             Show this help message
  --usage                                Display brief usage message
```

- USB CAN インターフェイスのシリアル番号を取得します：

```
$ pcan-settings -f=/dev/pcanusb32 -s
0x00000003
```

- USB2xCAN チャネルインターフェイスの CAN1 および CAN2 にデバイス番号 30 および 31 を設定します：

```
$ pcan-settings -f=/dev/pcanusb32 -d 30
$ pcan-settings -f=/dev/pcanusb33 -d 31
```

- USB2xCAN チャネルインターフェイスの CAN1 および CAN2 のデバイス番号を読取ります：



```
$ pcan-settings -f=/dev/pcanusb32 -d
30
$ pcan-settings -f=/dev/pcanusb33 -d
31
```

ドライバがリロードされると、これらの番号が読取られ、/sys にエクスポートされます。


```
$ cat /sys/class/pcan/pcanusb32/devid
30
$ cat /sys/class/pcan/pcanusb33/devid
31
```

したがって、Udev は通知を受け、ドライバのルールを読取ります。これらのデフォルトのルールでは、devid が-1 でない場合は、それを使用して、アダプタ名であるディレクトリの下に実際のデバイスノードへのシンボリックリンクを作成する必要があります。この例では、USB CAN インターフェイスが PCAN-USB Pro の場合、2 つのシンボリックリンクが /dev/pcan-usb_pro の下に作成されます。

```
$ ls -l /dev/pcan-usb_pro
total 0
drwxr-xr-x 2 root root 11 nov. 8 11:00 0
lrwxrwxrwx 1 root root 11 nov. 8 11:00 devid=30 -> ../pcanusb32
lrwxrwxrwx 1 root root 11 nov. 8 11:00 devid=31 -> ../pcanusb33
```

-  **注：**注：デバイス番号は、sysfs インターフェイスを使用して定義することもできます（21 ページの /sysfs インターフェイスを参照）。
-  **注：**v8.10 以降、デバイス番号は PCAN PCI Express / PCAN-PCIe FD カードの各 CAN チャネルに設定することもできます。

4.7.4 bitratetest

-  **注：**このアプリケーションは歴史的な理由で保持されますが、ビットレート値とクロック選択が新しい API によってユーザースペースに提案されるようになったため、非推奨となりました。

このアプリケーションは、よく知られたビットレート値の BTR0BTR1 値を表示します。BTR0BTR1 16 ビットコード化は、8 MHz SJA1000 コントローラ固有です。

使用法 :

```
$ bitratetest -help

bitratetest Version "Release_20150617_a" (www.peak-system.com)
----- Copyright (C) 2004-2009 PEAK System-Technik GmbH -----
bitratetest comes with ABSOLUTELY NO WARRANTY. This is free software and you are
welcome to redistribute it under certain conditions. For details see attached
COPYING file.

bitratetest - a small test the calculation of BTR0BTR1 data from PCAN.
usage:   bitratetest [-f=deviceinode] [-?]
         -f=deviceinode - path to devicefile, default=/dev/pcan0
         -? or --help   - this help

bitratetest: finished (0).
```

4.7.5 pcanfdtst

このアプリケーションは、ドライバによって処理されるすべてのデバイスノードとの間で CAN 2.0 / CAN FD メッセージを送受信できるため、ドライバのテストを可能にします。いくつかのモードで動作します :

- RX モードで実行している場合、アプリケーションは、すべての CAN FD デバイスノードから受信したすべてを画面に表示します。着信 CAN フレームの内容を確認することもできます
- TX モードで実行している場合、アプリケーションはすべてのデバイスへ CAN FD フレームを送信し、受信したイベントも表示します。

さらに、このアプリケーションは、RT Linux と non-RT Linux の両方でのドライバの新しい CAN FD API の使用法を示しています。アプリケーションは次のことを可能にします。

- CAN FD 使用の nominal ビットレートと Data ビットレートを指定します
- CAN FD モードの ISO および non-ISO を選択します
- マルチメッセージの transmit (送信) / receive (受信) を可能にする新しい API の新しいエントリポイントの使用法を示します
- 新しいイベントベースの API をデモンストレーションします
- いくつかのデバイス固有のオプション値を取得して設定します
- 記録されたファイルから CAN FD フレームを再生します

使用法：

```

$ pcanfdtst --help
Setup CAN[FD] tests between CAN channels over the pcan driver (>= v8.x)

WARNING
  This application comes with ABSOLUTELY NO WARRANTY. This is free software and
  you are welcome to redistribute it under certain conditions. For details, see
  attached COPYING file.

USAGE
  $ pcanfdtst MODE [OPTIONS] FILE [FILE...]

MODE
  Tx          generate CAN traffic on the specified CAN interfaces
  Rx          check CAN traffic received on the specified CAN interfaces
  Getopt      get a specific option value from the given CAN interface(s)
  Setopt      set an option value to the given CAN interface(s)
  Rec         same as 'tx' but frames are recorded into the given file

FILE
  For all modes except 'rec' mode:

  /dev/pcanx  indicate which CAN interface is used in the test.
               Several CAN interfaces can be specified. In that case,
               each one is opened in non-blocking mode.

  'rec' mode only:

  file_name   file path in which frames have to be recorded.

OPTIONS
  -a | --accept f-t      add message filter [f...t]
  -b | --bitrate v       set [nominal] bitrate to "v" bps
  --btr0btr1            bitrates with BTR0BTR1 format
  -B | --brs             data bitrate used for sending CANFD msgs
  -c | --clock v         select clock frequency "v" Hz
  -D | --debug           (maybe too) lot of display
  -d | --dbrate v        set data bitrate to "v" bps
  --dsample-pt v        define the data bitrate sample point ratio x 10000
  -E | --esi             set ESI bit in outgoing CANFD msgs
  --echo                tx frame is echoed by the hw into the rx path
  -f | --fd              select CAN-FD ISO mode
  --fd-non-iso          select CAN-FD non-ISO mode
  -F | --filler v|r|I    select how data are filled:
                        fixed value, randomly or incrementally
  -h | --help            display this help
  -i | --id v|r|I        set fixed CAN Id. to "v", randomly or incr.
  -is v|r|I              set fixed standard CAN Id "v", randomly or incr.
  -ie v|r|I              set fixed extended CAN Id "v", randomly or incr.
  -I | --incr v          "v"=nb of data bytes to use for increment counter
  -l | --len v|r|I       set fixed CAN dlc "v", randomly or incr.
  -m | --mul v           tx/rx "v" msgs at once
  -M | --max-duration v  define max duration the test should run in s.
  -n v                   send/read "v" CAN msgs then stop
  -o | --listen-only     set pcan device in listen-only mode
  --opt-name v           specify the option name (getopt/setopt modes)
  --opt-value v          specify the option value (getopt/setopt modes)
  --opt-size v           specify the option size (getopt/setopt modes)

```

```

-p | --pause-us v      "v" us. pause between sys calls (rx/tx def=0/1000)
  --play file         play recorded frames from "file" according to MODE
  --play-forever      file same as --play but loop forever on "file"
-P | --tx-pause-us v force a pause of "v" us. between each Tx frame
                      (if hw supports it)
-q | --quiet          nothing is displayed
-r | --rtr            set the RTR flag to msgs sent
  --no-rtr           clear the RTR flag from msgs sent
-s | --stdmsg-only    don't handle extended msgs
  --sample-pt v      define the bitrate sample point ratio x 10000
-t | --timeout-ms v   wait "v" ms. for events
-T | --check-ts       check host vs. driver timestatmps, stop if wrong
--ts-base v          set timestamp base [0..2]
--ts-mode v          set hw timestamp mode to v (hw dependant)
-u | --bus-load       get bus load notifications from the driver
-v | --verbose        things are (very much) explained
-w | --with-ts        logs are prefixed with time of day (s.us)
+FORMAT              output line format:
                      %t timestamp (s.us format)
                      %d direction (< or >)
                      %n device node name
                      %i CAN Id. (hex format)
                      %f flags
                      %l data length
                      %D data bytes
                      (default format is: "%t %n %d %i %f %l - %D")

```

- `opt-name`、`opt-value`、および `opt-size` パラメータは、`getopt` または `setopt` モードの場合にのみ使用されます。これらのオプションを使用すると、デバイスのグローバルまたは特定のオプション値を取得または設定できます (`int pcanfd_get_option ()` も参照)。
- ビットレートとクロック値は、係数 1,000 または係数 1,000,000 のショートカットとして `k` または `M` で表すことができます。オプション `--btr0btr1` が使用されている場合、ビットレートおよびビットレートオプションの値は `BTR0BTR1` フォーマットのコード化された値として読み取られることに注意してください。
- 各 `write` または `read` システムコール間の一時停止遅延の単位は μs です。ここで、値に追加された `m` (たとえば、`5m`) を使用すると、ミリ秒に変更され、`s` は秒 (たとえば、`7s`) に変更されます。
- `timeout-ms` パラメータの単位はミリ秒です。値を `s` にすると、秒に切り替わります (例 : `7s`)。
- コマンドラインで PC CAN インターフェイスが 1 つだけ指定されている場合、アプリケーションは “blocking” モードで実行されます。つまり、アプリケーションタスクはドライバの受信キューが空の間、または、ドライバの送信キューがいっぱいになっている間、ドライバにブロックされます。
- コマンドラインで複数の PC CAN インターフェイスが指定されている場合、アプリケーションは次のことを行います :
 - `non-blocking` で実行され、`non-RT` 環境で `select ()` システムコールを使用して、一度に複数のイベントを待機できるようにします。
 - 指定されたデバイスノードと同じ数のリアルタイムタスクを作成し、同時に複数のイベントを待機できるようにします。
- アプリケーションのデフォルトの動作は、ドライバとの間でメッセージを 1 つずつ `read / write` することです。`--mul x` オプションが使用されている場合 (`x > 1` の場合)、アプリケーションは `x` メッセージを一度に `reads / writes` します。

- +option は、Linux シェルコマンド "date" のように実行される文字列です：これにより、出力行の独自のフォーマットを指定できます。
- --ts-base オプションを使用すると、ユーザーはドライバが受信したフレームのタイムスタンプのベースを設定できます。

--ts-base	Description
0	タイムスタンプはホスト時間に基づいています (デフォルト)
1	タイムスタンプは、デバイスが開かれた時間に基づいています。
2	タイムスタンプは、ドライバがロードされた時間に基づいています。

- 一部のオプション (id、len、incr、filler...など) は、tx (または rec) モードまたは rx モードのいずれかで使用できます。
 - tx モードで使用すると、送信された CAN フレームの生成方法を制御します
 - rx モードで使用する場合、受信した CAN フレームの状態を制御します。

例：

1. 2 番目の USB CAN インターフェイスを使用して、ビットレート 250 kbit / s のバスに毎秒 10 個の CAN 2.0 フレーム (ランダム ID とデータ長) をライトします：

```
$ pcanfdtst tx -n 10 -b 250k -p 1s /dev/pcanusb33
0.429301518 /dev/pcanusb33 > BUS_STATE=ACTIVE [Rx:0 Tx:0]
0.4293989212 /dev/pcanusb33 < 567 ..... 7 - 00 00 00 00 00 00 00
1.4293989342 /dev/pcanusb33 < 069 ..... 7 - 00 00 00 00 00 00 00
2.4293989614 /dev/pcanusb33 < 451 ..... 7 - 00 00 00 00 00 00 00
3.4293989798 /dev/pcanusb33 < 44a ..... 3 - 00 00 00
4.4293989995 /dev/pcanusb33 < 729 ..... 1 - 00
5.4293990176 /dev/pcanusb33 < 0ba ..... 4 - 00 00 00 00
6.4293990468 /dev/pcanusb33 < 1f2 ..... 7 - 00 00 00 00 00 00 00
7.4293990660 /dev/pcanusb33 < 1e3 ..... 4 - 00 00 00 00
8.4293990845 /dev/pcanusb33 < 07c ..... 0 -
9.4293991023 /dev/pcanusb33 < 054 ..... 1 - 00
/dev/pcanusb33 < [packets=10 calls=10 bytes=41 eagain=0]
sent frames: 10
```

2. ホストの 1 番目の PCI インターフェイスとホストの 2 番目の USB インターフェイスで 60MHz クロックを使用します。:

拡張 ID0x123 および 24 データバイトの CAN FD (non-ISO) フレームを、nominal ビットレート 1 Mbit / s および Data ビットレート 2Mbit / s で書き込みます。

```
$ pcanfdtst tx --fd-non-iso -n 10 -ie 0x123 -l 24 -b 1M -d 2M -c 60M /dev/pcanusbfd33
/dev/pcanpcifd0
0.001871 /dev/pcanusbfd33 > BUS_STATE=ACTIVE [Rx:0 Tx:0]
0.022460 /dev/pcanusbfd33 < 00000123 .e... 24 - 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0.000000 /dev/pcanpcifd0 > BUS STATE=ACTIVE [Rx:0 Tx:0]
0.023558 /dev/pcanpcifd0 < 00000123 .e... 24 - 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0.024662 /dev/pcanusbfd33 < 00000123 .e... 24 - 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0.025754 /dev/pcanpcifd0 < 00000123 .e... 24 - 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

3. 前述と同じですが、フレームを "test.rec" という名前のファイルに（書込む代わりに）記録します：

```
$ pcanfdtst rec --fd-non-iso -n 10 -ie 0x123 -l 24 test.rec
0.022460 test.rec < 00000123 .e... 24 - 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0.023558 test.rec < 00000123 .e... 24 - 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0.024662 test.rec < 00000123 .e... 24 - 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0.025754 test.rec < 00000123 .e... 24 - 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

4. CAN バス（1Mbps）上の PCAN-PCIe FD の 1 番目のチャンネルを介して 1 秒ごとにフレームを書き込むファイル "test.rec" を再生します：

```
$ pcanfdtst tx --fd-non-iso --play test.rec -b 1M -p 1s /dev/pcanpcifd0
0.022460 /dev/pcanpcifd0 < 00000123 .e... 24 - 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
1.022460 /dev/pcanpcifd0 < 00000123 .e... 24 - 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
2.022460 /dev/pcanpcifd0 < 00000123 .e... 24 - 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3.022460 /dev/pcanpcifd0 < 00000123 .e... 24 - 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

5. 同じバスをリードしますが、最初の USB インターフェイスから：

```
$ pcanfdtst rx --fd-non-iso -b 1M -d 2M -c 60M /dev/pcanusbfd32
0.001848 /dev/pcanusb32 > BUS_STATE=ACTIVE [Rx:0 Tx:0]
14.761845 /dev/pcanusb32 > 00000123 .e... 24 - 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
14.764041 /dev/pcanusb32 > 00000123 .e... 24 - 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
14.766249 /dev/pcanusb32 > 00000123 .e... 24 - 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

6. フレームを送信しますが、multi-messages write API の新しいエントリーポイントを使用します。ここで、アプリケーションは同じフレームの 3 つのコピーを送信します。

```
$ /pcanfdtst tx --fd-non-iso -n 10 --mul 3 -ie 0x123 -I 4 -b 1M -d 2M -c 60M
/dev/pcanpcifd0
0.000000 /dev/pcanpcifd0 > BUS_STATE=ACTIVE [Rx:0 Tx:0]
0.000283 /dev/pcanpcifd0 < 00000123 .e... 4 - 00 00 00 00
0.001426 /dev/pcanpcifd0 < 00000123 .e... 4 - 01 00 00 00
0.002528 /dev/pcanpcifd0 < 00000123 .e... 4 - 02 00 00 00
0.003675 /dev/pcanpcifd0 < 00000123 .e... 4 - 03 00 00 00
0.005042 /dev/pcanpcifd0 < 00000123 .e... 4 - 04 00 00 00
0.006147 /dev/pcanpcifd0 < 00000123 .e... 4 - 05 00 00 00
0.007252 /dev/pcanpcifd0 < 00000123 .e... 4 - 06 00 00 00
0.008349 /dev/pcanpcifd0 < 00000123 .e... 4 - 07 00 00 00
0.009457 /dev/pcanpcifd0 < 00000123 .e... 4 - 08 00 00 00
0.010564 /dev/pcanpcifd0 < 00000123 .e... 4 - 09 00 00 00
/dev/pcanpcifd0 < [packets=30 calls=10 bytes=120 eagain=0]
sent frames: 30
```

同じバスで読取ると、ドライバが各フレームを 3 回書込んだことがわかります。

```
$ pcanfdtst rx --fd-non-iso -b 1M -d 2M -c 60M /dev/pcanusbfd32
0.001802 /dev/pcanusbfd32 > BUS STATE=ACTIVE [Rx:0 Tx:0]
8.714190 /dev/pcanusbfd32 > 00000123 .e... 4 - 00 00 00 00
8.714307 /dev/pcanusbfd32 > 00000123 .e... 4 - 00 00 00 00
8.714424 /dev/pcanusbfd32 > 00000123 .e... 4 - 00 00 00 00
8.714540 /dev/pcanusbfd32 > 00000123 .e... 4 - 01 00 00 00
8.714656 /dev/pcanusbfd32 > 00000123 .e... 4 - 01 00 00 00
8.714772 /dev/pcanusbfd32 > 00000123 .e... 4 - 01 00 00 00
8.715402 /dev/pcanusbfd32 > 00000123 .e... 4 - 02 00 00 00
8.715518 /dev/pcanusbfd32 > 00000123 .e... 4 - 02 00 00 00
8.715634 /dev/pcanusbfd32 > 00000123 .e... 4 - 02 00 00 00
8.716552 /dev/pcanusbfd32 > 00000123 .e... 4 - 03 00 00 00
8.716668 /dev/pcanusbfd32 > 00000123 .e... 4 - 03 00 00 00
...
```

注 : RT Linux の実行中、RT デバイスは “/dev” の下に表示されないため、RT バージョンの pcanfdtst は、CAN デバイスの実際の名前である “pcanX” を使用する**必要**があります。RT デバイスの作成時に Udev が作成できるエイリアスもリンクもありません。

7. getopt / setopt モードを使用して PCAN-USB FD デバイスのデバイス ID を変更する :

```
$ pcanfdtst getopt --opt-name 1 /dev/pcanusbfd32
/dev/pcanusbfd32 > [option=1 size=4 value=[ff ff ff ff ]
$ pcanfdtst setopt --opt-name 1 --opt-value 0xEFBEADDE --opt-size 4 /dev/pcanusbfd32
/dev/pcanusbfd32 > [option=1 size=4 value=[de ad be ef ]
$ pcanfdtst getopt --opt-name 1 /dev/pcanusbfd32
/dev/pcanusbfd32 > [option=1 size=4 value=[de ad be ef ]
```

8. PCAN-USB FD アダプタで実行されているファームウェアのバージョンを取得します。

```
$ pcanfdtst getopt --opt-name 11 /dev/pcanusbfd32
/dev/pcanusbfd32 > [option=11 size=4 value=[00 00 02 03 ]
```

4.8 netdev モード

Linux 用の PCAN ドライバが SocketCAN⁴ 用にビルドされている場合 (つまり、netdev モード)、SocketCAN コミュニティによって提案された CAN ユーティリティだけでなく一部のネットワークツールとの実行にも互換性があります。

注 : カーネルバージョン 3.6 以降、すべての PEAK-System PC CAN インターフェイスを備えた netdev インターフェイスは、メインラインカーネルにネイティブに含まれています。そのため、アプリケーションで SocketCAN インターフェイスを使用する場合は、Linux 用の PCAN ドライバをインストールする必要はありません。

このモードでは、ドライバは、列挙する PC CAN インターフェイスごとに “CAN network interface” を登録します。各ネットワークインターフェイスには、prefix can と、それに続く 0 から始まる番号で構成される名前が付けられます。

⁴ バックグラウンド情報 : <https://en.wikipedia.org/wiki/SocketCAN>

4.8.1 パラメータの割り当て

ドライバの割り当てパラメータ（11 ページの表 2：ドライバモジュールのパラメータで説明）を使用すると、デフォルトの昇順番号割り当てモデルを破ることができます。

assign = peak


パラメータ `assign = peak` を使用してドライバをロードすると、CAN ネットワークの CAN インターフェイス番号が PCAN デバイスのマイナー番号に固定されます。このモードでは、`canX` インターフェイスは `/dev/pcanX` と同じ PC CAN インターフェイスを定義します。

assign = pcanX : canY [, pcanX : canY]

パラメータ `assign = pcanX : canY` を使用してドライバをロードすると、名前 `canY` が `pcanX` という名前のデバイスに設定されます。このモードを選択する場合、割り当てパラメータ値は、それぞれが “,”（カンマ）で区切られた複数の割り当てのリストにすることができます。

assign = devid[,peak]

パラメータ `assign = devid` を使用してドライバをロードする場合、ネットワーク CAN インターフェイスの名前は、対応する PC CAN インターフェイスの `devid` 値を使用して作成されます。PC CAN インターフェイスで除算が定義されていない場合は、`assign = devid,peak` が使用されていない限り、通常の（昇順）順序列挙スキームが使用されます（`assign =` が使用されなかったかのように）。その場合、CAN ネットワーク番号は PCAN デバイス番号と同じになります（`assign = peak` が使用されたかのように）。

 **注：** `devid` プロパティの値は、`test / pcan-settings` ユーティリティを使用して変更できます（31 ページの 4.7.3 `pcan-settings` を参照）。

4.8.2 defclk パラメータ

ドライバの `defclk` パラメータ（11 ページの表 2：ドライバモジュールのパラメータで説明）を使用すると、CAN インターフェイスのデフォルトのクロック値を変更できます。一部の PEAK-System PC-CAN インターフェイスは、正確なビットタイミングを得るために、あるクロックから別のクロックに切替えるようにプログラムできます。

defclk = value

パラメータ `defclk = value` を指定してドライバをロードすると、すべての PC-CAN インターフェイスがデフォルトのクロック値から指定されたクロック値に切り替えようとします、`value`（値）は Hz で表されます。末尾の文字 “M” または “k” は、“x1000000” または “x1000” へのショートカットとして使用できます。例えば：

```
defclk = 12M
```

このようなクロックで実行できる各 PC CAN インターフェイスの 12MHz クロックを選択します。PC-CAN インターフェイスが指定されたクロック値を選択できない場合、PC-CAN インターフェイスはそれを無視します。`value`（値）が 0 の場合、デフォルトのクロックは変更されません。

defclk = pcanX : valueA [, pcanY : valueB]

パラメータ `defclk = pcanX : valueA [, pcanY : valueB]` を使用してドライバをロードすると、文字列で指定された名前が各 PC CAN インターフェイスに特定のクロック値を定義します。例えば：

```
defclk = pcan0 : 12M, pcan1 : 60M, pcan2 : 0, pcan3 : 24M
```

最初の 4 つの PC CAN インターフェイスに、それぞれ 12 MHz、60 MHz、デフォルト、および 24MHz クロックに切替えるように指示します。これらのインターフェイスのいずれかが特定のクロックを選択できない場合、それは黙って無視します。

4.8.3 ifconfig / iproute2

これらのユーティリティは両方とも canX インターフェイスを構成します。ifconfig は古いため、CAN / CAN FD 固有の機能のすべてをサポートできませんが、iproute2 パッケージの最新バージョン（特に ip ツール）には、canX インターフェイスをセットアップするためのオプションが含まれています。v8 以降、ドライバによってエクスポートされた canX インターフェイスは、ip link コマンドを使用して構成できます。



注： canX インターフェイスの構成には、root 権限が必要です。

ip ツールは、CAN および CAN FD のプロトコル固有の機能を処理するように変更されました。これにより、CAN インターフェイスのビットレート設定が簡素化されます。ツールのヘルプは、その使用法を説明しています。

```
$ ip link set can0 type can help
Usage: ip link set DEVICE type can
      [ bitrate BITRATE [ sample-point SAMPLE-POINT ] ] |
      [ tq TQ prop-seg PROP_SEG phase-seg1 PHASE-SEG1
        phase-seg2 PHASE-SEG2 [ sjw SJW ] ]

      [ loopback { on | off } ]
      [ listen-only { on | off } ]
      [ triple-sampling { on | off } ]
      [ one-shot { on | off } ]
      [ berr-reporting { on | off } ]

      [ restart-ms TIME-MS ]
      [ restart ]

Where: BITRATE      := { 1..1000000 }
       SAMPLE-POINT := { 0.000..0.999 }
       TQ           := { NUMBER }
       PROP-SEG     := { 1..8 }
       PHASE-SEG1   := { 1..8 }
       PHASE-SEG2   := { 1..8 }
       SJW          := { 1..4 }
       RESTART-MS   := { 0 | NUMBER }
```

次のいずれかのオプションを使用して、ビットレートを CAN インターフェイスに設定できるようになりました。

- bitrate ビットタイミングパラメータセット
(別名サンプルポイント、tq、prop-seg、phase-seg1、phase-seg2、sjw)
- bitrate オプションの後に数値が続く
(カーネルコンフィグレーションオプション CONFIG_CAN_CALC_BITTIMING が設定されている場合)

restart-ms オプションは、タイマーを ms 単位で定義します。この期間の後、CAN インターフェイスは BUS-OFF 状態で自動的に再起動されます。指定された数値が 0 の場合、自動再起動メカニズムは無効になっているため、ユーザーは手動で次の操作を行う必要があります：

```
$ sudo ip link set can0 type can restart
```

CAN ネットワークで IP リンクツールを使用する方法の最後のバージョンは、オンラインで入手できます。

<https://www.kernel.org/doc/Documentation/networking/can.txt>

例 :

500 kbit / s で PCAN `netdev` インターフェイスをセットアップします。

```
$ ip link set canX up type can bitrate 500000
```

1 Mbit / s の nominal ビットレートおよび 2Mbit / s の Data ビットレート(サポートされている場合)を使用して、PCAN `netdev` CAN FD インターフェイスをセットアップします。

```
$ ip link set canX up type can bitrate 1000000 dbitrate 2000000 fd on
```

non-ISOモードで実行される 1Mbit / s の nominal ビットレートと 2Mbit / s の Data ビットレートで PCAN `netdev` CAN FD インターフェイスをセットアップします (デバイスとカーネルでサポートされている場合)。

```
$ ip link set canX up type can bitrate 1000000 dbitrate 2000000 fd-non-iso on
```



注 : `iproute2` パッケージの最新バージョンは、次の場所からダウンロードできます。

<https://www.kernel.org/pub/linux/utils/net/iproute2/>

(`iproute2-ss141224 v3.18` は問題ありません)

インターフェイスを UP または DOWN に設定するためにのみ `ifconfig` を使用できます。

```
$ ifconfig canX down
# canX can't be used no more
$ ifconfig canX up
# canX can be used by any application
```



注 : ループバックモードは、ドライバの v8.10 以降でサポートされています。次の例は、送信された各フレームのエコーと、コントローラによってループバックされたフレームを受信するように `can0` を設定する方法を示しています。

```
$ sudo ip link set can0 up type can loopback on bitrate 500000
```

```
$ candump -x can0
```

```
$ cansend can0 123#0011223344556677
```

```
$ candump -x can0
Can0 TX - - 123 [8] 00 11 22 33 44 55 66 77
Can0 RX - - 123 [8] 00 11 22 33 44 55 66 77
```

4.8.4 can-utils

can-utils package⁵には、PCAN *netdev* インターフェイスを介した CAN および CAN FD メッセージの送受信を可能にするツールとユーティリティがいくつか含まれています。

注： SocketCAN ネットワークインターフェイスを介した CAN バスとの間の送受信では、これらのインターフェイスを構成する必要があります（40 ページの 4.8.3 *ifconfig* / *iproute2* を参照）。

例：

- canX インターフェイスから受信した CAN / CAN FD メッセージをダンプし、タイムスタンプを表示します：

```
$ candump -t a canX
```

- 4 データバイトの canX で ID 0x123 の CAN メッセージを送信する 0011 22 33：

```
$ cansend canX 123#00112233
```

- canX で CAN FD (##) を使用して同じメッセージを送信し、データバイトのデータビットレートを選択します (BRS フラグ= 1)：

```
$ cansend can1 123###100112233
```

4.9 USB 大容量ストレージデバイスモード

ドライババージョン 8.8 以降、特定の PC CAN インターフェイスを大容量記憶装置 (MSD) モードに切替えることができます。このモードでは、PC CAN インターフェイスはシステムの外付けディスクドライブとして表示されます。このモードの目的は、PC CAN インターフェイスのファームウェアのアップグレードを容易にすることです。そのモードになったら、その後通常モードで再起動するには、PC CAN インターフェイスをオフにする必要があります。

PC CAN インターフェイスは、そのデバイスノードが `/sysfs` ツリーの下に `mass_storage_mode` ファイルをエクスポートする場合、MSD モードに切替えることができます。以下の例では、PCAN-USB アダプタは MSD モードに切替えることができませんが、PCAN-USB FD は次のことができます：

```
$ cat /sys/class/pcan/pcanusb33/mass_storage_mode
cat: /sys/class/pcan/pcanusb33/mass_storage_mode: No such file or directory
$ cat /sys/class/pcan/pcanusbfd38/mass_storage_mode
0
```

注： `mass_storage_mode` ファイル (存在する場合) を読取ると、常に文字列 "0" が返されます。

MSD モードへの切替えは、PC CAN インターフェイスファームウェアをアップグレードする必要がある場合にのみ役立ちます。その場合、スーパーユーザーはまず PEAK-System Web サイトのサポートページから 互換性のある ファームウェアを入手する必要があります。

⁵ ウェブサイト can-utils : <https://github.com/linux-can/can-utils/>

ユーザーは、次の2つの方法でデバイスを MSD モードに切り替えることができます。

1. root 権限で、(たとえば) PC CAN インターフェイスの最初のデバイスノードに対応するディレクトリエントリの `mass_storage_mode` ファイルに 1 を書き込みます :

```
# echo 1 > /sys/class/pcan/pcanusbfd38/mass_storage_mode
```

sudo のユーザーは、代わりに以下のコマンドを入力する必要があります :

```
$ sudo sh -c "echo 1 > /sys/class/pcan/pcanusbfd38/mass_storage_mode"
```

2. root 権限で、`pcan-settings` テストアプリケーションを実行します :

```
$ sudo pcan-settings -M -f /dev/pcanusbfd38
pcan-settings: Mass Storage mode successfully set
Please wait for the LED(s) of the USB device to flash, then, if not
automatically done by the system, mount a VFAT filesystem on the newly
detected USB Mass Storage Device "/dev/sdX".
```



ヒント: “verbose” モードでは、次の手順に役立つ可能性のある詳細が表示されます :

```
$ sudo pcan-settings -v -M -f /dev/pcanusbfd38
pcan-settings: Mass Storage mode successfully set

The device node "/dev/pcanusbfd38" doesn't exist anymore.
Please wait for the LED(s) of the USB device to flash, then, if not
automatically done by the system, mount a VFAT filesystem on the newly
detected USB Mass Storage Device "/dev/sdX".

For example:
# mkdir -p /mnt/pcan-usb
# mount -t vfat /dev/sdX /mnt/pcan-usb
# ls -al /mnt/pcan-usb
```

数秒後、PC CAN インターフェイスの LED が点滅し、カーネルが新しい USB 大容量ストレージデバイスを検出します :

```
$ dmesg | tail -15
[27207.291209] usb 2-1.3.1: USB disconnect, device number 42
[27211.354058] usb 2-1.3.1: new high-speed USB device number 45 using ehci-pci
[27211.462592] usb 2-1.3.1: New USB device found, idVendor=0c72, idProduct=0101
[27211.462596] usb 2-1.3.1: New USB device strings: Mfr=0, Product=0, SerialNumber=0
[27211.462977] usb-storage 2-1.3.1:1.0: USB Mass Storage device detected
[27211.463223] scsi host11: usb-storage 2-1.3.1:1.0
[27212.482743] scsi 11:0:0:0: Direct-Access USB to CAN 1.0 PQ: 0 ANSI: 0 CCS
[27212.483167] sd 11:0:0:0: Attached scsi generic sg4 type 0
[27212.483718] sd 11:0:0:0: [sde] 2048 512-byte logical blocks: (1.05 MB/1.00 MiB)
[27212.484335] sd 11:0:0:0: [sde] Write Protect is off
[27212.484339] sd 11:0:0:0: [sde] Mode Sense: 03 00 00 00
[27212.484944] sd 11:0:0:0: [sde] No Caching mode page found
[27212.484951] sd 11:0:0:0: [sde] Assuming drive cache: write through
[27212.490199] sde:
[27212.492690] sd 11:0:0:0: [sde] Attached SCSI removable disk
```

前述の例では、カーネルはストレージデバイス `sde` を `idVendor 0c72` で新しく検出された USB 大容量ストレージデバイスに接続しました。実行中の Linux システムがそのストレージデバイスにファイルシステムを自動的にマウントしない場合、スーパーユーザーは手動でマウントする必要があります。

1. マウントポイントを作成します（たとえば、`/mnt/pcan-usb-fd`）：

```
$ sudo mkdir -p /mnt/pcan-usb-fd
```

2. ストレージデバイス全体をそのマウントポイントにマウントします：


```
$ sudo mount -t vfat /dev/sde /mnt/pcan-usb-fd
```

3. マウントされたデバイスの内容を確認します（例）：

```
$ ls -al /mnt/pcan-usb-fd
total 830
drwxr-xr-x 2 root root 512 janv. 1 1970 .
drwxr-xr-x 10 root root 4096 déc. 13 11:23 ..
-rwxr-xr-x 1 root root 844800 avril 1 2015 firmware.bin
```

4. 既存のファームウェアファイルを削除します：

```
$ sudo rm -f /mnt/pcan-usb-fd/*.bin
```

 **注：** 削除操作は仮想ですが、ストレージデバイスが新しいファームウェアファイルを保存するのに十分な大きさであることをシステムに認識させるために必須です。その時点で、PC CAN インターフェイスが接続されていない場合、再度接続されると、通常どおり再起動します。


5. 新しいファームウェアファイルをコピーします：

```
$ sudo cp PCAN-New_firmware_file.bin /mnt/pcan-usb-fd
```

6. ストレージデバイスのすべてのマウントポイントをアンマウントします：

```
$ sudo umount -A /dev/sdX
```

数秒後、新しいファームウェアを実行するには、PC CAN インターフェイスの電源を入れ直す必要があります。PC CAN インターフェイスが USB ケーブル以外の電源（たとえば、PCAN-USB X6 など）から電力を供給されている場合は、プラグを抜くか、スイッチをオフにします。

 **注：** PCAN-USB X6 アダプタには3つのモジュールが装備されており、それぞれが2つの CAN ポートを管理します。また、各モジュール（CAN1、CAN3、CAN5）の最初のデバイスノードを毎回使用して、前の操作を合計3回実行する必要があります。

たとえば、接続された PCAN-USB X6 アダプタが次のようなシステムによってエクスポートされた場合：

```
$ lspcan -t -T -i
dev name port irq clock btrs bus
[PCAN-USB X6 0]
|_ pcanusbfd38 CAN1 - 80MHz 500k+2M CLOSED
|_ pcanusbfd39 CAN2 - 80MHz 500k+2M CLOSED
|_ pcanusbfd40 CAN3 - 80MHz 500k+2M CLOSED
|_ pcanusbfd41 CAN4 - 80MHz 500k+2M CLOSED
|_ pcanusbfd42 CAN5 - 80MHz 500k+2M CLOSED
|_ pcanusbfd43 CAN6 - 80MHz 500k+2M CLOSED
```

それから、pcanusbfd38、pcanusbfd40、及び pcanusbfd42 をすべて MSD モードに切替える必要があります。

再起動すると、PC CAN インターフェイスは新しいファームウェアを実行します。PC CAN インターフェイス（存在する場合）に組み込まれているファームウェアのバージョンは、 /sysfs ツリーで読取ることができます。

例えば：

```
$ cat /sys/class/pcan/pcanusbfd38/adapter_version
3.2.0
```

5 開発者ガイド

7 ページの「3.1 バイナリのビルド」で説明されているように、Linux 用の PCAN ドライバは、次の 2 つの占有モードで実行するように構成できます：

1. *chardev* モードでビルドされている場合、ドライバは従来の `open / read / write / ioctl / close` の character デバイスインターフェイスをユーザースペースアプリケーションにエクスポートします。
2. *netdev* モードでビルドされている場合、ドライバはソケットインターフェイスをエクスポートします。



注： リアルタイム環境用のドライバをビルドする場合、*netdev* モードは使用できません。

7 ページの「3.1 バイナリのビルド」および 9 ページの「3.2 パッケージのインストール」で説明されているようにドライバをビルドおよびインストールすると、ドライバへのシステムコールをカプセル化するいくつかのユーザー API ライブラリもビルドおよびインストールされます。

- `libpcan` は古い API であり、常に CAN 2.0 チャンネルへのアクセスを提供しています (47 ページの 5.1.1 CAN 2.0 API を参照)。
- `libpcanfd` は、ドライバのバージョン 8 以降のパッケージに含まれている新しい API です。この新しい API は、CAN 2.0 および CAN FD チャンネルへのアクセス、およびマルチメッセージサービスとステータスイベントメッセージングを提供します。このライブラリには、47 ページの 5.1.1 CAN 2.0 API で説明されている `libpcan` のすべてのエントリポイントも含まれているため、このライブラリは、`libpcan` を使用する代わりに CAN 2.0 API アプリケーションとリンクすることもできます。

これらのライブラリは両方とも、リアルタイムアプリケーションで使用するために構築できます。これらのライブラリを構築する場合、2 つの RT 環境を選択できます。

▶ **Xenomai** リアルタイムタスクを実行するためのリアルタイムライブラリを構築するには：

```
$ make -C lib RT=XENOMAI # Or "make xeno" since pcan 8.2
```

▶ **RTAI** リアルタイムタスクを実行するためのリアルタイムライブラリを構築するには：

```
$ make -C lib RT=RTAI # Or "make rtai" since pcan 8.2
```

5.1 chardev モード

このモードでは、Linux 用の PCAN ドライバは、CAN / CAN FD チャンネルごとに 1 つのデバイスノードを作成し、検出してマイナー番号を接続します (ドライバに固有)。すべての character モードドライバと同様に、Linux 用の PCAN ドライバはシステムによってメジャー番号が付けられています。

各デバイスノードは、開いたり、閉じたり、読取り、書込みを行うことができます (4.6 `read / write` インターフェイス (26 ページ) を参照)。主な関数は、`ioctl ()` エントリポイントを介して実装されます。v8 以降のドライバパッケージのいくつかのソフトウェアコンポーネントのアーキテクチャは、次ページの図 1 に要約されています。

`chardev` モードは、Windows システムと Linux システムの両方で実行できるアプリケーションを作成するために、PEAK-System が開発した PCAN-Basic API を利用したい場合に必要です。

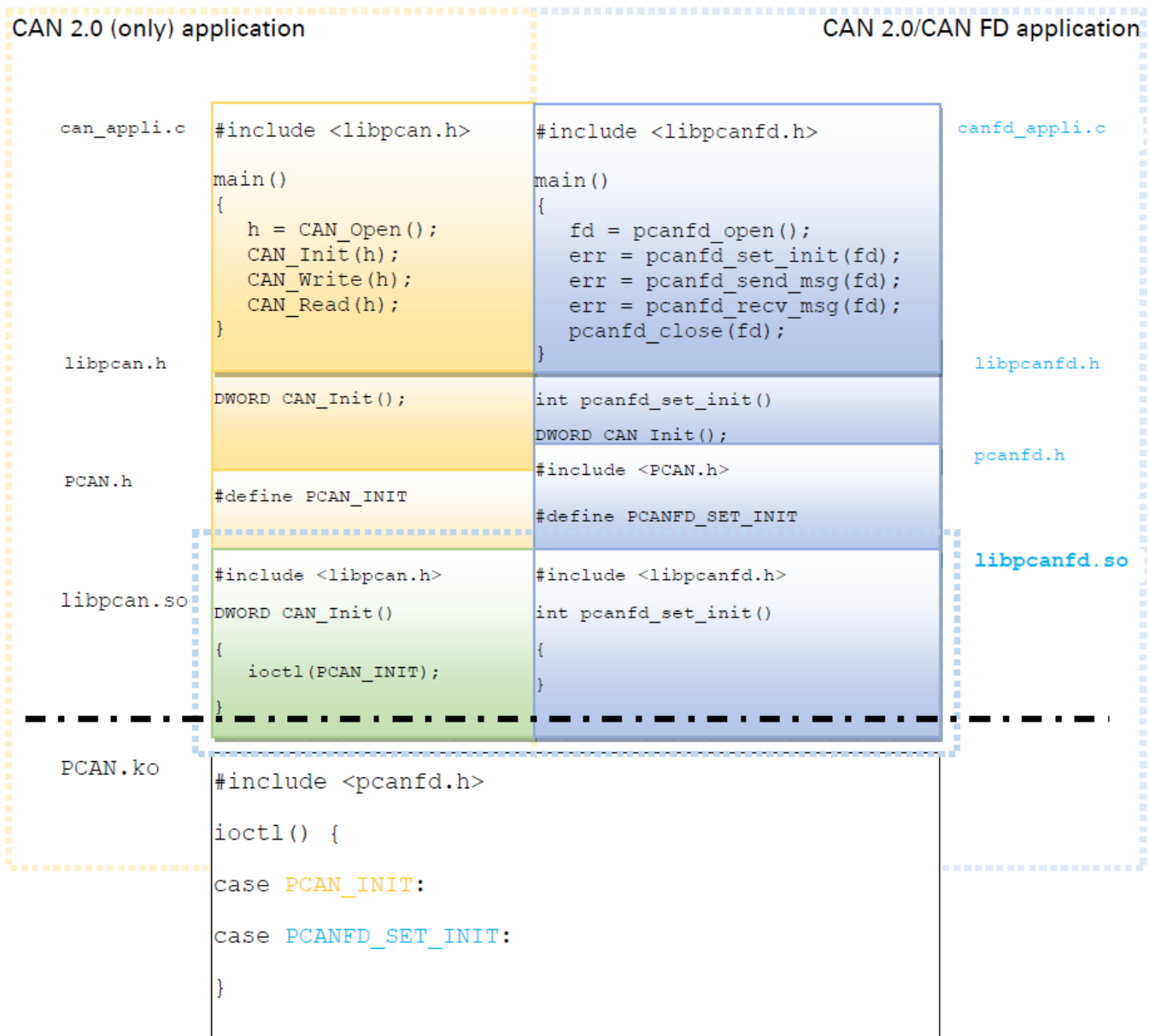


図 1: ソフトウェアコンポーネントアーキテクチャ

5.1.1 CAN 2.0 API

注: この API は下位互換性の理由で保持されているため、これらのエントリポイントも新しい `libpcanfd` ライブラリによって提案されます。ただし、この API は非推奨と見なされます。代わりに、新しい CAN FD API を使用してください。

(古い) CAN 2.0 API `ioctl` コードは、`pcan.h` によって定義されています。

```
#define PCAN_INIT          _IOWR(PCAN_MAGIC_NUMBER, MYSEQ_START, TPCANInit)
#define PCAN_WRITE_MSG    _IOW (PCAN_MAGIC_NUMBER, MYSEQ_START + 1, TPCANMsg)
#define PCAN_READ_MSG     _IOR (PCAN_MAGIC_NUMBER, MYSEQ_START + 2, TPCANRdMsg)
#define PCAN_GET_STATUS   _IOR (PCAN_MAGIC_NUMBER, MYSEQ_START + 3, TPSTATUS)
#define PCAN_DIAG         _IOR (PCAN_MAGIC_NUMBER, MYSEQ_START + 4, TPDIAG)
#define PCAN_BTROBTR1     _IOWR(PCAN_MAGIC_NUMBER, MYSEQ_START + 5, TPBTROBTR1)
#define PCAN_GET_EXT_STATUS _IOR (PCAN_MAGIC_NUMBER, MYSEQ_START + 6, TPEXTENDEDSTATUS)
#define PCAN_MSG_FILTER    _IOW (PCAN_MAGIC_NUMBER, MYSEQ_START + 7, TPMSGFILTER)
#define PCAN_EXTRA_PARAMS _IOWR(PCAN_MAGIC_NUMBER, MYSEQ_START + 8, TPEXTRAPARAMS)
```

この API を使用すると、PEAK-System の任意の PC CAN インターフェイスとの間で CAN 2.0 メッセージ（のみ）を read / write できます。

この API は、libpcan ライブラリによってカプセル化されます（テストディレクトリに格納されている transmitest、receivetest、bitratetest、pcan-settings などの C / C ++ プログラムはこの API を使用します）。この API は CAN 2.0 アクセスで常にサポートされているため、この API を使用するには、アプリケーションが -lpcan または -lpcanfd とリンクする必要があります。

この API の原則は、CAN バスへの接続の全期間中に使用されるオブジェクト HANDLE のようなもので CAN 2.0 チャンネルを実装することです。この API は、Windows®用の PCAN-Light バージョンに大きく影響を受けています。

ライブラリは、次のエントリポイントを定義します：

HANDLE CAN_Open (WORD wHardwareType、 ...) ;

この関数は、タイプ (PCI、USB、ISA...) とチャンネル番号（または選択したタイプに応じて他の引数）に応じて CAN 2.0 チャンネルを開きます。wHardwareType でサポートされている値のリストを取得するには、pcan.h で定義されている HW_XXX シンボルのリストを参照してください。

例えば：

```
#include <libpcan.h>

/* open the 2nd CAN 2.0 PCI channel in the system (first is 0) */
HANDLE h = CAN_Open(HW_PCI, 1);
```

DWORD CAN_Init (HANDLE hHandle、WORD wBTR0BTR1、int nCANMsgType) ;

この関数は、開かれた CAN 2.0 チャンネルをビットレート (BTR0BTR1 SJA1000 フォーマットで表される) および CAN メッセージの extended ID に設定された（または設定されていない）フィルタで初期化します。

wBTR0BTR1 値および nCANMsgType でサポートされている値のリストを取得するには、libpcan.h で定義されている CAN_BAUD_XXX および CAN_INIT_TYPE_XX シンボルのリストを参照してください。

例えば：

```
#include <libpcan.h>

/* CAN 2.0 channel handle */
HANDLE h;
DWORD status;

...

/* initialize the CAN 2.0 channel with 500 kbps BTR0BTR1, accepting extended ID. */
status = CAN_Init(h, CAN_BAUD_500K, CAN_INIT_TYPE_EX);
```


DWORD CAN_Write (HANDLE hHandle, TPCANMsg * pMsgBuff) ;

この関数は、開いた CAN 2.0 チャンネルを介して CAN 2.0 メッセージを CAN バスにライトします。

例えば :

```
#include <libpcan.h>

/* CAN 2.0 channel handle */
HANDLE h;
DWORD status;
TPCANMsg msg;

...
msg.ID = 0x123
msg.MSGTYPE = MSGTYPE_STANDARD;
msg.LEN = 3;
msg.DATA[0] = 0x01;
msg.DATA[1] = 0x02;
msg.DATA[2] = 0x03;

/* write standard msg ID = 0x123. with 3 data bytes 0x01 0x02 0x03
 * (the function may block)
 */
status = CAN_Write(h, &msg);
```

DWORD CAN_Read (HANDLE hHandle, TPCANMsg * pMsgBuff) ;

この関数は、開いた CAN 2.0 チャンネルを介して CAN バスから受信した CAN 2.0 メッセージをリードします。メッセージが受信されていない場合、呼び出し元のタスクはブロックされます。

例えば :

```
#include <libpcan.h>

/* CAN 2.0 channel handle */
HANDLE h;
DWORD status;
TPCANMsg msg;

...
/* wait for a CAN 2.0 msg received from the CAN channel
 * (the function may block)
 */
status = CAN_Read(h, &msg);
```

DWORD CAN_Status (HANDLE hHandle) ;

この関数は、開いている CAN 2.0 チャンネルのステータスを返します (cat / proc / pcan で表示される最後の列に対応します)。戻り値はビットマスクです (各ビットの意味を取得するには、 pcan.h で定義されている CAN_ERR_xxx シンボルのリストを参照してください)。



注： この機能でチャンネルのステータスを読取るとクリアされます！

例えば :

```
#include <libpcan.h>

/* CAN 2.0 channel handle */
HANDLE h;
DWORD status;

...

/* get the status of a CAN 2.0 channel */
status = CAN_Status(h);
```

DWORD CAN_Close (HANDLE hHandle) ;

この関数は、開いている CAN 2.0 チャンネルを閉じます。指定されたハンドルは次に使用しないでください。

例えば :

```
#include <libpcan.h>

/* CAN 2.0 channel handle */
HANDLE h;

...

/* wait for a CAN 2.0 msg received from the CAN channel
 * (the function may block)
 */
CAN_Close(h);
```

Linux の multi-tasking 環境に役立つために、ライブラリは次の LINUX_XXX () 関数で拡張されています。

int LINUX_CAN_FileHandle (HANDLE hHandle) ;

この関数は、ドライバによって開かれたデバイスノードに対応するファイル記述子を返します。これは、アプリケーションが複数の read / write イベントを待機する必要がある場合に役立ちます。

HANDLE LINUX_CAN_Open (const char * szDeviceName、 int nFlag) ;

この関数は CAN 2.0 チャンネルを開きますが、代わりに Linux システムデバイスノード名を使用します。

DWORD LINUX_CAN_Read (HANDLE hHandle、 TPCANRdMsg * pMsgBuff) ;

この関数は「DWORDCAN_Read (HANDLE hHandle、 TPCANMsg * pMsgBuff) ;」のように機能しますが、追加のタイムスタンプ情報を返します。

DWORD LINUX_CAN_Read_Timeout (HANDLE hHandle、 TPCANRdMsg * pMsgBuff、 int nMicroSeconds) ;

この関数は「DWORDLINUX_CAN_Read (HANDLE hHandle、 TPCANRdMsg * pMsgBuff) ;」のように機能しますが、CAN からリードメッセージがない場合は、nMicroSeconds の呼び出しタスクをブロックします。

DWORD LINUX_CAN_Write_Timeout (HANDLE hHandle、 TPCANMsg * pMsgBuff、 int nMicroSeconds) ;

この関数は「DWORDCAN_Write (HANDLE hHandle、 TPCANMsg * pMsgBuff) ;」のように機能しますが、CAN チャンネルの送信キューに空きがない場合、nMicroSeconds の呼び出しタスクをブロックします。

DWORD LINUX_CAN_Extended_Status (HANDLE hHandle、 int * nPendingReads、 int * nPendingWrites) ;

この関数は「DWORDCAN_Status (HANDLE hHandle) ;」のように機能しますが、* nPendingReads 内のチャンネルの受信キューからのリードを待機しているメッセージの数、そして、* nPendingWrites 内のチャンネルの送信キューから送信されるのを待機しているメッセージの数を返します。

DWORD LINUX_CAN_Statistics (HANDLE hHandle、TPDIAG * diag) ;

この関数は、CAN 2.0 チャンネルに関するいくつかの統計を提供しますが、このチャンネルのステータスをクリアすることはありません (「DWORDCAN_Status (HANDLEhHandle) ;」のように)。

WORD LINUX_CAN_BTR0BTR1 (HANDLE hHandle、DWORD dwBitRate) ;

この関数は、指定されたビットレートに対応する BTR0BTR1 8 MHz SJA1000 コードを返します。

5.1.2 CAN FD API

この API は、ドライバのバージョン 8 以降の新機能です。古いもので定義されたエントリポイントとデータ構造を常に提案しますが (47 ページの 5.1.1 CAN 2.0 API を参照)、いくつかの新しいデータ構造と ioctl コードの定義を追加します (pcanfd.h を参照)。古いエントリポイントでは通常どおり CAN 2.0 バスに接続できますが、新しいエントリポイントでは CAN 2.0 および CAN FD バスに接続できます。言い換えれば、新しい API は、CAN バスにアクセスするための新しい最新のユニバーサルな方法です。古いエントリポイントは、既存のアプリケーションコードとの下位互換性を確保するためにのみ保持されます。

```
#define PCANFD_SET_INIT      _IOW(PCAN_MAGIC_NUMBER, PCANFD_SEQ_SET_INIT, ¥
                             struct pcanfd_init)
#define PCANFD_GET_INIT     _IOR(PCAN_MAGIC_NUMBER, PCANFD_SEQ_GET_INIT, ¥
                             struct pcanfd_init)
#define PCANFD_GET_STATE    _IOR(PCAN_MAGIC_NUMBER, PCANFD_SEQ_GET_STATE, ¥
                             struct pcanfd_state)
#define PCANFD_ADD_FILTERS  _IOW(PCAN_MAGIC_NUMBER, PCANFD_SEQ_ADD_FILTERS, ¥
                             struct pcanfd_msg_filters)
#define PCANFD_GET_FILTERS  _IOW(PCAN_MAGIC_NUMBER, PCANFD_SEQ_GET_FILTERS, ¥
                             struct pcanfd_msg_filters)
#define PCANFD_SEND_MSG     _IOW(PCAN_MAGIC_NUMBER, PCANFD_SEQ_SEND_MSG, ¥
                             struct pcanfd_msg)
#define PCANFD_RECV_MSG     _IOR(PCAN_MAGIC_NUMBER, PCANFD_SEQ_RECV_MSG, ¥
                             struct pcanfd_msg)
#define PCANFD_SEND_MSGS    _IOWR(PCAN_MAGIC_NUMBER, PCANFD_SEQ_SEND_MSGS, ¥
                             struct pcanfd_msgs)
#define PCANFD_RECV_MSGS    _IOWR(PCAN_MAGIC_NUMBER, PCANFD_SEQ_RECV_MSGS, ¥
                             struct pcanfd_msgs)
#define PCANFD_GET_AVAILABLE_CLOCKS _IOWR(PCAN_MAGIC_NUMBER, ¥
                                           PCANFD_SEQ_GET_AVAILABLE_CLOCKS, ¥
                                           struct pcanfd_available_clocks)

#define PCANFD_GET_BITTIMING_RANGES _IOWR(PCAN_MAGIC_NUMBER, ¥
                                           PCANFD_SEQ_GET_BITTIMING_RANGES, ¥
                                           struct pcanfd_bittiming_ranges)
#define PCANFD_GET_OPTION    _IOWR(PCAN_MAGIC_NUMBER, PCANFD_SEQ_GET_OPTION, ¥
                             struct pcanfd_option)
#define PCANFD_SET_OPTION    _IOW(PCAN_MAGIC_NUMBER, PCANFD_SEQ_SET_OPTION, ¥
                             struct pcanfd_option)
```

これらの新しい `ioctl` コードは、新しい `libpcanfd` ライブラリのいくつかの新しいエントリポイントによってもカプセル化されます。これらの新しいエントリポイントは、`libpcanfd.h` で定義されています。



注： test アプリケーション `pcanfdtst` は、これらの新しいエントリポイントを使用します。

この新しいライブラリは、CAN チャンネルを `HANDLE` オブジェクトにカプセル化することはなくなりましたが、対応するデバイスノードで行われた `open ()` システムコールによって返されるファイル記述子を直接処理します。



注： 古い API と新しい API には 互換性はありません。CAN チャンネルが 1 つの API を介して開かれると、他の API と一緒に使用することはできません。つまり、CAN チャンネルを開くと、接続に使用される API が選択されます。

新しい API は、いくつかのレベルの使用法を提供します。レベル 1 は前記の `ioctl` コードをカプセル化しますが、レベル 2 API は、デバイスノードを開いたり閉じたりするためのより使いやすい方法を提供します。

最後に、この新しい API のすべてのエントリポイントは整数値を返します。負の場合は、`-errno` に等しいエラーコードとして解釈する必要があります。

int pcanfd_set_init (int fd, struct pcanfd_init * pfdi) ;

この関数は、CAN 2.0 および CAN FD プロパティ（対応するハードウェアに互換性がある場合）を選択できるようにするいくつかの新しい設定で、開いているデバイスノードを初期化します。これらのプロパティは、新しい `struct pcanfd_init` オブジェクトによって定義されます（`pcanfd.h` も参照）。

```

struct pcanfd_init {
    __u32 flags;
    __u32 clock_Hz;
    struct pcan_bittiming nominal;
    struct pcan_bittiming data;
};

```

Field	Values	Description
flags	PCANFD_INIT_LISTEN_ONLY	デバイスはリスッソ専用モードで開かれます。
	PCANFD_INIT_STD_MSG_ONLY	標準の CAN メッセージ ID のみが送受信されます。設定されていない場合、そのデバイスにはすべての種類のメッセージ ID が使用されます。
	PCANFD_INIT_FD	デバイスが CAN-FD 対応の場合は、CAN FD ISO アクセス用にデバイスを開きます。
	PCANFD_INIT_FD_NON_ISO	デバイスが CAN-FD 対応の場合は、CAN FD non-ISO アクセス用にデバイスを開きます。
	PCANFD_INIT_TS_DEV_REL	CAN バスから受信したメッセージに対してドライバが設定するタイムスタンプは、デバイスが初期化された瞬間を基準にしています。
	PCANFD_INIT_TS_HOST_REL	CAN バスから受信したメッセージに対してドライバが設定するタイムスタンプは、ホストが起動した瞬間を基準にしています。
	PCANFD_INIT_TS_DRV_REL	CAN バスから受信したメッセージに対してドライバが設定するタイムスタンプは、ドライバが起動した瞬間を基準にしています（デフォルト）。
	PCANFD_INIT_BUS_LOAD_INFO	CAN バス負荷が対応するハードウェアが提供できる情報である場合、ドライバは定期的にこのチャンネルの rx fifo キューに STATUS メッセージを入れて、チャンネルが接続されている現在のバス負荷をアプリケーションに通知します。
clock	0	CAN デバイスのデフォルトのクロックは、ドライバによって選択されます（デフォルト）。
	any other value	適切なビットタイミング仕様を選択するために CAN デバイスハードウェアで選択するクロック周波数（Hz で表される）。
nominal	struct pcan_bittiming	CAN バスへの接続に使用する nominal ビットレートを定義します（デフォルト値は前記の表 2 で定義されています）。
data	struct pcan_bittiming	デバイスが CAN FD デバイスであり、書き込みメッセージフラグ PCANFD_MSG_BRS が設定されている場合に選択するデータビットレートを定義します（デフォルト値は前記の表 2 で定義されています）。

表 9： structpcanfd_init の説明

int pcanfd_get_init (int fd、 struct pcanfd_init * pfdi) ;

この関数を使用すると、ユーザーアプリケーションは、開いているデバイスに設定されている初期化設定を取得できます。

int pcanfd_get_state (int fd、 struct pcanfd_state * pfdi) ;

この関数は、開いているデバイスの現在の状態を取得します。CAN チャンネルの状態は、新しい struct pcanfd_state オブジェクトに要約されます（ pcanfd.h も参照）：

```

struct pcanfd_state {
    __u16 ver_major, ver_minor, ver_subminor;

    struct timeval tv_init;          /* time the device was initialized */

    enum pcanfd_status bus_state;    /* CAN bus state */

    __u32 device_id;                /* device ID, ffffffff is unused */

    __u32 open_counter;             /* open() counter */
    __u32 filters_counter;          /* count of message filters */

    __u16 hw_type;                  /* PCAN device type */
    __u16 channel_number;           /* channel number for the device */

    __u16 can_status;               /* same as wCANStatus but NOT CLEARED */
    __u16 bus_load;                 /* bus load value, ffff if not given */

    __u32 tx_max_msgs;              /* Tx fifo size in count of msgs */
    __u32 tx_pending_msgs;          /* msgs waiting to be sent */
    __u32 rx_max_msgs;              /* Rx fifo size in count of msgs */
    __u32 rx_pending_msgs;          /* msgs waiting to be read */
    __u32 tx_frames_counter;        /* Tx frames written on device */
    __u32 rx_frames_counter;        /* Rx frames read on device */
    __u32 tx_error_counter;         /* CAN Tx errors counter */
    __u32 rx_error_counter;         /* CAN Rx errors counter */

    __u64 host_time_ns;             /* host time in nanoseconds as it was */
    __u64 hw_time_ns;              /* when hw_time_ns has been received */
};

```

int pcanfd_add_filter (int fd, struct pcanfd_msg_filter * pf) ;

この関数は、デバイスのフィルタリストにメッセージフィルタを追加します。デバイスが開かれると、デバイスのフィルタは存在しません。つまり、アプリケーションは CAN バスから読取られたすべてのメッセージ ID を受信します。メッセージフィルタを追加すると、アプリケーションに渡され、破棄される着信 CAN メッセージをフィルターリングできます。メッセージフィルタは、新しい struct pcanfd_msg_filter オブジェクト (pcanfd.h も参照) によって記述されます :

```

struct pcanfd_msg_filter {
    __u32 id_from;                  /* msgs ID in range [id_from..id_to] */
    __u32 id_to;                   /* and flags == msg_flags */
    __u32 msg_flags;               /* will be passed to applications */
};

```

int pcanfd_add_filters (int fd, struct pcanfd_msg_filters * pfl) ;

この関数は、複数のメッセージフィルタをデバイスのフィルタリストに一度に追加します。メッセージのリストは、次の struct pcanfd_msg_filters に保存されます。

```

struct pcanfd_msg_filters {
    __u32 count
    struct pcanfd_msg_filter list[0];
};

```

i **注：** count フィールドには、list[] 配列フィールドに保存されているメッセージフィルタの数が含まれている必要があります。

int pcanfd_add_filters_list (int fd, int count, struct pcanfd_msg_filter * pf) ;

この関数は、複数のメッセージフィルタをデバイスのフィルタリストに一度に追加します。

これは、 "int pcanfd_add_filters (int fd, struct pcanfd_msg_filters * pfl) ;" よりも使いやすいショートカットです。

int pcanfd_del_filters (int fd) ;

この関数は、デバイスのフィルタリストにリンクされているすべてのフィルタを削除します。デバイスのフィルタはもう存在しないため、アプリケーションは CAN バスから読取られたすべてのメッセージ ID を受信します。これは、CAN デバイスが開かれたときのデフォルトの動作です。

int pcanfd_send_msg (int fd, struct pcanfd_msg * pfdm) ;

この関数は、開いているデバイスを介して CAN バスにメッセージを書込みます。メッセージは、新しい struct pcanfd_msg オブジェクトによって定義されます (pcanfd.h も参照) :

```

struct pcanfd_msg {
    __u16          type;          /* PCANFD_TYPE_xxx */
    __u16          data_len;     /* true length (not the DLC) */
    __u32          id;
    __u32          flags;       /* PCANFD_xxx definitions */
    struct timeval timestamp;
    __u8           ctrlr_data[PCANFD_CTRLR_MAXDATALEN];
    __u8           data[PCANFD_MAXDATALEN] __attribute__((aligned(8)));
};

```

この C ストラクチャ・オブジェクトは、CAN 2.0 および CAN FD メッセージを伝送できます。また、ドライバがアプリケーションにプッシュできる out-of-band メッセージタイプ (ステータスメッセージなど) を含めることもできます。

i **注：** デバイスノードがノンブロックモードで開かれていない限り、CAN バスにメッセージを書込むと、呼出し元のタスクがブロックされる可能性があります。その場合、タスクに発信メッセージを格納するのに十分なスペースがなかった場合、この関数によって - EWOULDBLOCK が返されます。

Field	Values	Description
type	PCANFD_TYPE_CAN20_MSG	このメッセージはCAN 2.0メッセージです (data_lenフィールドは8より大きくすることはできません)。
	PCANFD_TYPE_CANFD_MSG	このメッセージはCAN FDメッセージです。PCANFD_MSG_BRSのようなビットは、flagsフィールドによって処理されます。data_lenフィールドは64より大きくすることはできません。
data_len	<= 8	CANバスで送信するためにデータフィールドからコピーするデータバイト数。
	<= 64	CAN FDメッセージの場合、この値は実際に書き込むバイト数です。ドライバは、これに対応するDLCコードに適合させる責任があります。
flags	PCANFD_MSG_RTR	Remote Transmission Request (リモート送信要求) メッセージ
	PCANFD_MSG_EXT	メッセージIDは29ビットを使用してコード化されます (標準のメッセージ形式は11ビットのみを使用します)。
	PCANFD_MSG_SLF	サポートされている場合、このメッセージはハードウェアによって内部受信キューにループバックされます。
	PCANFD_MSG_SNG	サポートされている場合、このメッセージはSingle-Shotモードで送信されます。つまり、CANフレームが正常に送信されない場合、それ以上の送信は試行されません。
	PCANFD_MSG_BRS	CAN FDの場合、このビットは、nominalビットレートの代わりに、データバイトの転送ビットレートスイッチ (Bit Rate Switch) を有効にします。
	PCANFD_MSG_ECHO	サポートされている場合、このメッセージはバスに書き込まれ、ハードウェアによって内部受信キューにエコーされます。
id		CANメッセージのID。
data		CANメッセージのデータバイト。data_lenフィールドで指定されたバイト数のみがバスにコピーされます。

表 10 : 送信側での structpcanfd_msg の使用法

int pcanfd_send_msgs (int fd、struct pcanfd_msgs * pfdml) ;

この関数は、開いているデバイスを介してメッセージのリストを CAN バスに書込みます。メッセージリストは、新しい struct pcanfd_msgs オブジェクトによって定義されます (pcanfd.h も参照)。

```

struct pcanfd_msgs {
    __u32          count;
    struct pcanfd_msg list[0];
};

```

この C 構造体オブジェクトは、いくつかの CAN 2.0 および CAN FD メッセージを転送できます。書込むメッセージの数は、count フィールドで指定されます。このフィールドは、デバイスの送信キューに実際に書込まれたメッセージの数を示すためにも使用されます。

i **注:** デバイスノードがノンブロックモードで開かれていない限り、CAN バスに複数のメッセージを書込むと、呼出しタスクがブロックされる可能性があります。その場合、タスクに送信メッセージを格納するのに十分なスペースがなかった場合、この関数によって -EWOULDBLOCK が返されます。

少なくとも 1 つのメッセージが送信キューに正常に書込まれた場合、関数は 0 を返します。それ以外の場合は、負のエラーコードを返します。

この関数を使用すると、メモリコピーが節約され、カーネルスペースとユーザースペース間の一定のラウンドトリップが節約されます。

例 :

```

#include <malloc.h>
#include <libpcanfd.h>

int fill_msg(struct pcanfd_msg *pm);

struct pcanfd_msgs *pml;
/* allocate enough room to store 5 CAN messages */
pml = malloc(sizeof(*pml) + 5 * sizeof(struct pcanfd_msg));
pml->count = 5;

for (pml->count = 0; pml->count < 5; pml->count++) {
    fill_msg(pml->list + pml->count);
}

/* put all of the messages at once in the transmit queue of the device... */
err = pcanfd_send_msgs(fd, pml);
if (err)
    printf("Only %u/5 msgs have been sent because of errno=%d\n",
           pml->count, err);

free(pml);
...

```

int pcanfd_send_msgs_list (int fd、int count、struct pcanfd_msg * pfdm) ;

この関数は、開いているデバイスを介してメッセージのリストを CAN バスに書込みます。

これは、 "int pcanfd_send_msgs (int fd、struct pcanfd_msgs * pfdml) ;" よりも使いやすいショートカットです。

int pcanfd_recv_msg (int fd、struct pcanfd_msg * pfdm) ;

この関数は、ドライバが対応するデバイスの受信キューにプッシュした可能性のある保留中のメッセージを読み取ります。このメッセージは、CAN バスから受信した CAN 2.0 または CAN FD メッセージが含まれている場合は帯域内メッセージ、ステータスメッセージが含まれている場合は out-of-band メッセージになります。

注：デバイスノードがノンブロックモードで開かれていない限り、ドライバからメッセージを読取ると、呼出し元のタスクがブロックされる可能性があります。その場合、タスクが読取るメッセージを見つけられなかった場合、この関数によって-EWOULDBLOCK が返されます。

Field	Values	Description
type	PCANFD_TYPE_CAN20_MSG	このメッセージは CAN 2.0 メッセージです。
	PCANFD_TYPE_CANFD_MSG	このメッセージは CAN FD メッセージです。PCANFD_MSG_BRS や PCANFD_MSG_ESI などのビットもフラグフィールドに設定できます。
	PCANFD_TYPE_STATUS	このメッセージはステータス メッセージであり、CAN デバイスの状態に関する詳細情報を提供します。
	PCANFD_TYPE_ERROR_MSG	このメッセージは、CAN バスから読み取られたエラーメッセージです。この種のメッセージはデフォルトでは受信されません (int pcanfd_set_option (int fd, int name, void * value, int size) ;のオプション PCANFD_ALLOWED_MSG_ERROR も参照してください) 。
data_len		CAN デバイスから受信したメッセージのデータバイト数。 CAN FD の場合、この値は送信側で指定された値と同じでない場合があることに注意してください。
id		CAN メッセージの ID
flags	PCANFD_MSG_RTR	Remote Transmission Request (リモート送信要求) メッセージ
	PCANFD_MSG_EXT	メッセージ ID の形式は拡張形式です。
	PCANFD_MSG_SLF	このメッセージは、ハードウェアによって内部受信キューに looped-back (ループバック) されています。
	PCANFD_MSG_SNG	このメッセージは Single-Shot (シングルショット) モードで送信されました。
	PCANFD_MSG_BRS	CAN FD の場合、このビットは、受信したメッセージのデータバイトを送信するためにデータビットレートが選択されていることを示します。
	PCANFD_MSG_ECHO	このメッセージは、ハードウェアによって内部受信キューにエコーされるだけでなく、バスに書き込まれます。
	PCANFD_MSG_ESI	CAN FD エラー インジケータ: CAN バスでエラーが検出されました。
	PCANFD_TIMESTAMP	timestamp (タイムスタンプ) フィールドは、メッセージが受信されたタイムスタンプで評価されます。
	PCANFD_HWTIMESTAMP	PCANFD_TIMESTAMP が設定されている場合、このフラグは、指定されたタイムスタンプがハードウェアによって指定されたタイムスタンプから作成されていることを示します。設定されていない場合、タイムスタンプはホスト時間からドライバによって作成されています。
	PCANFD_ERRCNT	ctrlr_data[PCANFD_RXERRCNT] および ctrlr_data[PCANFD_TXERRCNT] には、CAN コントローラから読み取られた Rx および Tx エラー カウンターが含まれます。
PCANFD_BUSLOAD	ctrlr_data [PCANFD_BUSLOAD_UNIT]には、ハードウェアコントローラによって計算されたバス負荷のパーセンテージが含まれ、ctrlr_data [PCANFD_BUSLOAD_DEC]には、小数部分が含まれます。	
timestamp	struct timeval	フラグフィールドに PCANFD_TIMESTAMP が設定されている場合、これはメッセージが受信された瞬間を示します。PCANFD_HWTIMESTAMP も設定されている場合、指定されたモーメントはハードウェアアキュロックから作成された時間です。PCANFD_HWTIMESTAMP が設定されていない場合、この瞬間は、ホストの現在の時刻からドライバによって作成されます (int pcanfd_set_option (int fd, int name, void * value, int size) ;のオプションも参照してください) 。
ctrlr_data		CAN コントローラ固有のデータ (上記の PCANFD_ERRCNT および PCANFD_BUSLOAD フラグを参照)。
data		CAN メッセージのデータバイト。受信したデータバイトの数は、data_len フィールドで指定されます。


表 11：受信側での structpcanfd_msg の使用法

```
int pcanfd_recv_msgs (int fd, struct pcanfd_msgs * pfdml) ;
```

この機能は、ドライバデバイスの受信キューからメッセージのリストを一度に読み取ることができます。メッセージリストは、新しい struct pcanfd_msgs オブジェクトによって定義されます (pcanfd.h も参照)。

```
struct pcanfd_msgs {
    __u32          count;
    struct pcanfd_msg list[0];
};
```

この C 構造体オブジェクトは、いくつかの CAN 2.0 および CAN FD メッセージを伝送できます。リストに含めることができるメッセージの最大数は、count フィールドで設定する必要があります。この関数から戻ると、count フィールドは、コピーされたメッセージの実際の数にドライバによって設定されます。

 **注：** デバイスノードがノンブロックモードで開かれていない限り、ドライバからいくつかのメッセージを読取ると、呼出し元のタスクがブロックされる可能性があります。その場合、タスクが読取るメッセージを見つけられなかった場合、この関数によって-EWOULDBLOCK が返されます。

少なくとも 1 つのメッセージが正常に読み取られた場合、関数は 0 を返します。それ以外の場合、負のエラーコードを返します。

この関数を使用すると、メモリコピーが節約され、カーネルスペースとユーザースペース間の一定のラウンドトリップが節約されます。

例：

```

#include <malloc.h>
#include <libpcanfd.h>
#include <errno.h>

int process_msg(struct pcanfd_msg *pm)
{
    switch (pm->type) {
        case PCANFD_TYPE_CAN20_MSG:
            return process_CAN_2_0_msg(pm);
        case PCANFD_TYPE_CANFD_MSG:
            return process_CAN_FD_msg(pm);
        case PCANFD_TYPE_STATUS:
            return process_status_msg(pm);
        case PCANFD_TYPE_ERROR_MSG:
            /* if enabled, see PCANFD_OPT_ALLOWED_MSGS[PCANFD_ALLOWED_MSG_ERROR] */
            return process_error_msg(pm);
    }
    return -EINVAL;
}

struct pcanfd_msgs *pml;
int i, err;

/* allocate enough room to store at least 5 CAN messages */
pml = malloc(sizeof(*pml) + 5 * sizeof(struct pcanfd_msg));
pml->count = 5;

/* waiting for these messages... */
err = pcanfd_recv_msgs(fd, pml);
if (err)
    exit(1);

/* process the received messages... */
for (i = 0; i < pml->count; i++) {
    process_msg(pml->list + i);
}

free(pml);
...

```

int pcanfd_recv_msgs_list (int fd, int count, struct pcanfd_msg * pm) ;

この機能は、ドライバデバイスの受信キューからメッセージのリストを一度に読取ることができます。

これは、"int pcanfd_recv_msgs (int fd, struct pcanfd_msgs * pfdml) ;" よりも使いやすいショートカットです。

戻り値が正の場合、デバイス入力キューから読取られたメッセージの実際の数を示します。それ以外の場合は、エラーコードです。

int pcanfd_get_available_clocks (int fd, struct pcanfd_available_clocks * pac) ;

この関数は、CAN / CAN FD デバイスで実行できるすべての使用可能なクロックのリストを返します。クロックは、デバイスが初期化されるときに選択されます (int pcanfd_set_init (int fd, struct pcanfd_init * pfdi) ;を参照)。

```

/* Device available clocks value */
struct pcanfd_available_clock {
    __u32    clock_Hz;
    __u32    clock_src;
};

struct pcanfd_available_clocks {
    __u32    count;
    struct pcanfd_available_clock list[0];
};

```

ユーザーは、“list []” 配列に割当てられたアイテムの数を使用して “count” フィールドを設定する必要があります。

例：

```

struct pcanfd_available_clocks *pac;
int i, err;


/* allocate enough room to store at least 8 clock values */
pac = malloc(sizeof(*pac) + 6 * sizeof(struct pcanfd_available_clock));
pac->count = 6;

/* reading the available clocks list */
err = pcanfd_get_available_clocks(fd, pac);
if (err)
    exit(1);

/* display all available clocks */
for (i = 0; i < pac->count; i++) {
    printf("clock #%u/%u: %u Hz\n", i, pac->count, pac->list[i]);
}

free(pac);

```

 **注：** list [0] には、常にデフォルトのクロック値が含まれています。CAN FD デバイスのみが複数のクロックを定義します。

int pcanfd_get_bittiming_ranges (int fd、struct pcanfd_bittiming_ranges * pbtr)

この関数は、CAN / CAN FD デバイスで実行できるすべての使用可能なビットタイミング範囲のリストを返します。ビットタイミングは、デバイスが初期化されるときに選択されます (int pcanfd_set_init (int fd、struct pcanfd_init * pfdi) を参照)。

```

/* CAN FD bittiming capabilities */
struct pcanfd_bittiming_range {
    __u32   brp_min;
    __u32   brp_max;
    __u32   brp_inc;
    __u32   tseg1_min;
    __u32   tseg1_max;
    __u32   tseg2_min;
    __u32   tseg2_max;
    __u32   sjw_min;
    __u32   sjw_max;
};

struct pcanfd_bittiming_ranges {
    __u32   count;
    struct pcanfd_bittiming_range list[0];
};

```

ユーザーは、"list[]" 配列に割当てられたアイテムの数を使用して "count" フィールドを設定する必要があります。

pcan ドライバのバージョン 8.2 は、CAN 2.0 デバイスの "count" フィールドに常に 1 を設定し、CAN FD デバイスの場合は 2 を設定します。

例：

```

struct pcanfd_bittiming_ranges *pbr;
int err;

/* allocate enough room to store 2 ranges */
pbr = malloc(sizeof(*pbr) + 2 * sizeof(struct pcanfd_bittiming_range));
pbr->count = 2;

/* reading the bit timings ranges list */
err = pcanfd_get_bittiming_ranges(fd, pbr);
if (err)
    exit(1);

if (pbr->count == 1)
    printf("CAN 2.0 device\n");
else
    printf("CAN FD device\n");

free(pbr);

```

int pcanfd_get_option (int fd, int name, void * value, int size) ;

この機能により、チャンネルデバイスに接続されているオプションの現在値を読取ることができます。各チャンネルは、開かれると値が初期化される同じオプションのセットを処理します。これらのオプションのリストを以下に示します。時間の経過とともに進化する可能性があります (pcanfd.h も参照)。

存在しないオプションの値を取得すると-EINVAL が返され、サポートされていないオプション (デバイス用) を取得すると-EOPNOTSUPP が返されます。値バッファが小さすぎるオプションの値を読取ると、-ENOSPC が返されます。

オプションの値を正常に読取ると、value (値) にコピーされたバイト数が返されます。

Option	Size	Description										
PCANFD_OPT_CHANNEL_FEATURES	4	<p>このオプションの値は、開いているチャンネルの機能を提供するビットマスクです。</p> <table border="0"> <tr> <td>PCANFD_FEATURE_FD</td> <td>チャンネルは CAN-FD 対応です</td> </tr> <tr> <td>PCANFD_FEATURE_IFRAME_DELAYUS</td> <td>フレーム間に遅延を挿入できます</td> </tr> <tr> <td>PCANFD_FEATURE_BUSLOAD</td> <td>チャンネルはバス負荷を計算できます</td> </tr> <tr> <td>PCANFD_FEATURE_HWTIMESTAMP</td> <td>タイムスタンプはデバイスから読み取られます</td> </tr> <tr> <td>PCANFD_FEATURE_DEVICEID</td> <td>チャンネルには、ユーザーデバイス ID でラベルを付けることができます。</td> </tr> </table>	PCANFD_FEATURE_FD	チャンネルは CAN-FD 対応です	PCANFD_FEATURE_IFRAME_DELAYUS	フレーム間に遅延を挿入できます	PCANFD_FEATURE_BUSLOAD	チャンネルはバス負荷を計算できます	PCANFD_FEATURE_HWTIMESTAMP	タイムスタンプはデバイスから読み取られます	PCANFD_FEATURE_DEVICEID	チャンネルには、ユーザーデバイス ID でラベルを付けることができます。
PCANFD_FEATURE_FD	チャンネルは CAN-FD 対応です											
PCANFD_FEATURE_IFRAME_DELAYUS	フレーム間に遅延を挿入できます											
PCANFD_FEATURE_BUSLOAD	チャンネルはバス負荷を計算できます											
PCANFD_FEATURE_HWTIMESTAMP	タイムスタンプはデバイスから読み取られます											
PCANFD_FEATURE_DEVICEID	チャンネルには、ユーザーデバイス ID でラベルを付けることができます。											
PCANFD_OPT_DEVICE_ID	4	チャンネルデバイスに接続されているユーザーIDを取得します (チャンネルでサポートされている場合)										
PCANFD_OPT_AVAILABLE_CLOCKS		<p>チャンネルデバイスで使用可能なクロックのリストを返します 返される値のタイプは <code>pcanfd_available_clocks</code> です (<code>pcanfd.h</code> および <code>int pcanfd_get_available_clocks (int fd, struct pcanfd_available_clocks * pac)</code> ;を参照してください)。</p> <p>このオプションを取得することは、<code>pcanfd_get_available_clocks ()</code> を呼び出すことと同じです。</p>										
PCANFD_OPT_BITTIMING_RANGES		<p>nominal ビットレートを指定するために、チャンネルで使用可能なビットタイミング範囲を指定します。これらの範囲は、チャンネルに装備されている CAN / CAN FD コントローラによって異なります (<code>int pcanfd_get_bittiming_ranges (int fd, struct pcanfd_bittiming_ranges * pbtr</code> も参照)。</p>										
PCANFD_OPT_DBITTIMING_RANGES		<p>Data ビットレートを指定するために、チャンネルで使用可能なビットタイミング範囲を指定します。これらの範囲は、チャンネルに装備されている CAN-FD コントローラによって異なります (<code>int pcanfd_get_bittiming_ranges (int fd, struct pcanfd_bittiming_ranges * pbtr</code> も参照)。</p>										
PCANFD_OPT_ALLOWED_MSGS	4	<p>このオプションの値は、アプリケーションが受信できるメッセージの種類を説明するビットマスクです。</p> <table border="0"> <tr> <td>PCANFD_ALLOWED_MSG_CAN</td> <td>CAN / CAN FD フレーム</td> </tr> <tr> <td>PCANFD_ALLOWED_MSG_RTR</td> <td>RTR フレーム</td> </tr> <tr> <td>PCANFD_ALLOWED_MSG_EXT</td> <td>Extended (拡張) Id.</td> </tr> <tr> <td>PCANFD_ALLOWED_MSG_STATUS</td> <td>STATUS メッセージ</td> </tr> <tr> <td>PCANFD_ALLOWED_MSG_ERROR</td> <td>CAN バスからのエラー</td> </tr> </table>	PCANFD_ALLOWED_MSG_CAN	CAN / CAN FD フレーム	PCANFD_ALLOWED_MSG_RTR	RTR フレーム	PCANFD_ALLOWED_MSG_EXT	Extended (拡張) Id.	PCANFD_ALLOWED_MSG_STATUS	STATUS メッセージ	PCANFD_ALLOWED_MSG_ERROR	CAN バスからのエラー
PCANFD_ALLOWED_MSG_CAN	CAN / CAN FD フレーム											
PCANFD_ALLOWED_MSG_RTR	RTR フレーム											
PCANFD_ALLOWED_MSG_EXT	Extended (拡張) Id.											
PCANFD_ALLOWED_MSG_STATUS	STATUS メッセージ											
PCANFD_ALLOWED_MSG_ERROR	CAN バスからのエラー											
PCANFD_OPT_ACC_FILTER_11B	8	チャンネルで受信された標準メッセージの現在の受け入れフィルタコードとマスクを取得します。上位 32 ビットには受け入れコードが含まれ、下位 32 ビットには受け入れマスクが含まれます。										
PCANFD_OPT_ACC_FILTER_29B	8	チャンネルで受信された拡張メッセージの現在の受け入れフィルタコードとマスクを取得します。上位 32 ビットには受け入れコードが含まれ、下位 32 ビットには受け入れマスクが含まれます。										
PCANFD_OPT_IFRAME_DELAYUS	4	送信する各フレーム間に CAN コントローラによって現在挿入されている遅延の値を μs で取得します。										
PCANFD_OPT_HWTIMESTAMP_MODE	4	<p>各構造体 <code>pcanfd_msg</code> に保存されたタイムスタンプを計算するためにドライバが現在使用している現在のモードを取得します。</p> <table border="0"> <tr> <td>PCANFD_OPT_HWTIMESTAMP_OFF</td> <td>ホスト時間のみに基づいています (hw が対応している場合でも)。</td> </tr> <tr> <td>PCANFD_OPT_HWTIMESTAMP_ON</td> <td>ホストタイムベース+Raw のハードウェアタイムオフセット。</td> </tr> <tr> <td>PCANFD_OPT_HWTIMESTAMP_COOKED</td> <td>ホストタイムベース+ Cooked ハードウェアタイムオフセット。</td> </tr> <tr> <td>PCANFD_OPT_HWTIMESTAMP_RAW</td> <td>Raw ハードウェアタイムスタンプ。</td> </tr> </table> <p>Cooked タイムスタンプは、異なるクロック・システム (PC、ボード、USB コントローラなど) 間のクロック・ドリフトを処理します。</p> <p>Raw ハードウェアタイムスタンプは、コントローラによって指定された 64 ビットの μs タイムスタンプであり、<code>s + μs</code> に変換されます。これらのタイムスタンプは、ホスト時間に関連していません。</p>	PCANFD_OPT_HWTIMESTAMP_OFF	ホスト時間のみに基づいています (hw が対応している場合でも)。	PCANFD_OPT_HWTIMESTAMP_ON	ホストタイムベース+Raw のハードウェアタイムオフセット。	PCANFD_OPT_HWTIMESTAMP_COOKED	ホストタイムベース+ Cooked ハードウェアタイムオフセット。	PCANFD_OPT_HWTIMESTAMP_RAW	Raw ハードウェアタイムスタンプ。		
PCANFD_OPT_HWTIMESTAMP_OFF	ホスト時間のみに基づいています (hw が対応している場合でも)。											
PCANFD_OPT_HWTIMESTAMP_ON	ホストタイムベース+Raw のハードウェアタイムオフセット。											
PCANFD_OPT_HWTIMESTAMP_COOKED	ホストタイムベース+ Cooked ハードウェアタイムオフセット。											
PCANFD_OPT_HWTIMESTAMP_RAW	Raw ハードウェアタイムスタンプ。											

Option	Size	Description
PCANFD_OPT_DRV_VERSION	4	ドライバのバージョンを取得します。
PCANFD_OPT_FW_VERSION	4	デバイスのファームウェアバージョンを取得します。
PCANFD_IO_DIGITAL_CFG	4	PCAN チップのデジタル I/O ピンの構成を取得/設定します (ファームウェア >= 3.3.0) : 0 I/O ピンは出力モードでセットアップされます。 1 I/O ピンは入力モードでセットアップされます。
PCANFD_IO_DIGITAL_VAL	4	デジタル I/O ピンの値を取得/設定します。
PCANFD_IO_DIGITAL_SET	4	デジタル I/O ピンを High に設定します。
PCANFD_IO_DIGITAL_CLR	4	デジタル I/O ピンを Low にクリアします。
PCANFD_IO_ANALOG_VAL	4	PCAN チップからアナログ I/O 値を取得します。
PCANFD_OPT_MASS_STORAGE_MODE	4	デバイスが大容量ストレージデバイスモードに切り替えることができる場合、このオプションの値は常に 0 です。 デバイスが MSD モードに切り替えることができない場合、このオプションの読み取りは失敗し、erno は EOPNOTSUPP に設定されます。
PCANFD_OPT_FLASH_LED	4	デバイスの LED を点滅させて識別します (デバイスに LED が装備されている場合)。 値は、LED が点滅するミリ秒数です。
PCANFD_OPT_DRV_CLK_REF	4	ドライバが使用するクロックリファレンスを取得します (21 ページの表 6 を参照)。

表 12

int pcanfd_set_option (int fd、int name、void * value、int size) ;

この機能により、チャンネルデバイスに接続されているオプションに値を設定できます。各チャンネルは、開かれると値が初期化される同じオプションのセットを処理します。変更可能なオプションのリストを以下に示します。時間の経過とともに進化する可能性があります (pcanfd.h も参照)。

存在しないオプションの値を設定するか、既存のオプションに無効な値を設定すると-EINVAL が返され、サポートされていないオプション (デバイスの場合) に値を設定すると-EOPNOTSUPP が返されます。

オプションに value (値) を正しく設定すると、0 が返されます。

Option	Size	Description
PCANFD_OPT_DEVICE_ID	4	チャンネルデバイスにユーザー数値を設定します (チャンネルでサポートされている場合)
PCANFD_OPT_ALLOWED_MSGS	4	アプリケーションに通知するメッセージの種類を設定します。開くと、各チャンネルは以下を受信できます。 PCANFD_ALLOWED_MSG_CAN CAN / CAN FD フレーム PCANFD_ALLOWED_MSG_RTR RTR フレーム PCANFD_ALLOWED_MSG_EXT Extended (拡張) Id. PCANFD_ALLOWED_MSG_STATUS STATUS メッセージ
PCANFD_OPT_ACC_FILTER_11B	8	チャンネルで受信される標準メッセージの現在の受け入れフィルタコードとマスクを設定します。上位 32 ビットには受け入れコードが含まれ、下位 32 ビットには受け入れマスクが含まれている必要があります。
PCANFD_OPT_ACC_FILTER_29B	8	チャンネルで受信した拡張メッセージの現在の受け入れフィルタコードとマスクを設定します。上位 32 ビットには受け入れコードが含まれ、下位 32 ビットには受け入れマスクが含まれている必要があります。
PCANFD_OPT_IFRAME_DELAYUS	4	CAN コントローラが送信できる場合は、CAN コントローラが送信する各フレーム間に挿入する必要がある遅延の値を μ s 単位で設定します。

Option	Size	Description
PCANFD_OPT_HWTIMESTAMP_MODE	4	<p>各 struct pcanfd_msg に保存されたタイムスタンプを計算するためにドライバが使用する必要がある現在のモードを設定します。</p> <p>PCANFD_OPT_HWTIMESTAMP_OFF ホスト時間のみに基づいています (hw が対応している場合でも)。</p> <p>PCANFD_OPT_HWTIMESTAMP_ON ホストタイムベース+Raw ハードウェアタイムオフセット。</p> <p>PCANFD_OPT_HWTIMESTAMP_COOKED ホストタイムベース+ Cooked ハードウェアタイムオフセット。</p> <p>PCANFD_OPT_HWTIMESTAMP_RAW Raw ハードウェアタイムスタンプ。</p> <p>Cooked タイムスタンプは、異なるクロック・システム (PC、ボード、USB コントローラなど) 間のクロック・ドリフトを処理します。</p> <p>Raw ハードウェアタイムスタンプは、コントローラによって指定された 64 ビットの μs タイムスタンプであり、s +μs に変換されます。これらのタイムスタンプは、ホスト時間に関連していません。</p>
PCANFD_OPT_MASS_STORAGE_MODE	4	<p>デバイスに互換性がある場合、このオプションに 0 以外の値を設定すると、PC CAN インターフェイスが大容量ストレージデバイスモードに切り替わります。</p> <p>デバイスが MSD モードに切り替えることができない場合、このオプションの設定は失敗し、ermo は EOPNOTSUPP に設定されます。</p> <p>ユーザーが root 権限を持っていない場合、このオプションの設定は失敗し、ermo は EPERM に設定されます。</p>

表 13

int pcanfd_open (char * dev_pcan, __u32 flags, ...);

この関数は、PC CAN インターフェイスを開いて初期化するために使用されるショートカットです。最初のパラメータは、システムが認識しているデバイスノードの名前です。2 番目の引数は、関数の次のパラメータとそのシーケンス順序、および CAN コントローラの初期化に使用される PCANFD_INIT_xxx フラグを示すビットマスクです (libpcanfd.h および pcanfd.h も参照)。

Bit	Description
OFD_BITRATE	<p>nominal ビットレートの指定は、3 番目のパラメータから始まります。</p> <p>OFD_BTR0BTR1 も設定されている場合、3 番目のパラメータは BTR0BTR1 SJA1000 フォーマットに関する 16 ビット値として解釈されます。</p> <p>OFD_BRPTSEGSJW が指定されている場合、3 番目、4 番目、5 番目、および 6 番目のパラメータは BRP、TSEG1、TSEG2、および SJW 値として解釈されます。</p> <p>上記のビットのいずれも設定されていない場合、3 番目の引数はビット/秒の値として解釈されます。</p>
OFD_SAMPLEPT	<p>nominal ビットレートの横の引数は、要求された最小サンプルポイントレートです。指定しない場合、ドライバは独自のデフォルト値を使用します。指定する場合、この値は 1/10000 で表す必要があります (つまり、8750 は 87.5%を表します)</p>
OFD_DBITRATE	<p>Data ビットレートは次の引数で与えられます:</p> <p>OFD_BTR0BTR1 も設定されている場合、次のパラメータは BTR0BTR1 SJA1000 フォーマットを尊重する 16 ビット値として解釈されます。</p> <p>OFD_BRPTSEGSJW が指定されている場合、次の 4 つのパラメータは、BRP、TSEG1、TSEG2、および SJW の値として解釈されます。</p> <p>上記のビットのいずれも設定されていない場合、次の引数はビット/秒の値として解釈されます。</p>
OFD_DSAMPLEPT	<p>Data ビットレートの横の引数は、要求された最小サンプルポイントレートです。指定しない場合、ドライバは独自のデフォルト値を使用します。指定する場合、この値は 1/10000 で表す必要があります (つまり、8750 は 87.5%を表します)</p>
OFD_CLOCKHZ	CAN コントローラで選択するクロック周波数 (Hz) は、次の引数で指定されます。
OFD_NONBLOCKING	デバイスノードはノンブロッキングモードで開かれます。
PCANFD_INIT_xxx	これらのフラグはすべて、"int pcanfd_set_init (int fd, struct pcanfd_init * pfdi);" を使用して初期化されたかのように、CAN デバイスを初期化するために使用されます。

表 14 : pcanfd_open () の flags 引数の使用法

例：

```
#include <libpcanfd.h>

int fd;

/* open the 1st CANFD channel of the PCAN-USB Pro FD and set 1Mbps+2Mbps bitrate */
fd = pcanfd_open("/dev/pcanusbprofd0",
                OFD_BITRATE|OFD_DBITRATE,
                1000000,
                2000000);

...
```

5.2 netdev モード

Linux 用の PCAN ドライバは、*netdev* モードで構築されています、つまり、次のようになります：

```
$ make -C driver NET=NETDEV_SUPPORT
```

または、

```
$ make -C driver netdev
```

この場合、ユーザーアプリケーションは *libpcan* ライブラリも *libpcanfd* ライブラリも使用できませんが、代わりにソケット API を介して構築する必要があります。プログラマーは、たとえば次のリンクから始めて、オンラインドキュメントにアクセスできます。

- <https://en.wikipedia.org/wiki/SocketCAN>
- <https://www.kernel.org/doc/Documentation/networking/can.txt>