X2CAN API
(verze 2.x)

Ing. David Španěl
Info@canlab.cz
www.canlab.cz

Obsah:

1. Úvod

V souvislosti s vývojem interface USB2CAN bylo vytvořeno nové API, které umožňuje jednotně používat jak převodník PP2CAN, tak nový, moderní převodník USB2CAN. Struktura tohoto API dovoluje jednoduše začlenit i případné další varianty CAN interface karet, včetně karet třetích výrobců. API zastřešuje základní funkce interface PP2CAN, USB2CAN a virtuálního portu V2CAN tak, aby se sjednotila volání základních funkcí pro otevření, komunikaci a uzavření portu interface. Zdrojový kód cílové aplikace pak není závislý na použitém adaptéru. Přesto zůstává dále otevřená cesta pro použití různých "specialit" daného interface.

2. Obecné funkce, základní datové struktury

X2CAN API došlo k unifikaci definice struktury nesoucí CAN zprávu. Tato struktura je definována takto:

```
typedef struct
{
    unsigned short Hour;
    unsigned short Minute;
    unsigned short Second;
    unsigned short Milliseconds;
} CANMessageTime;

typedef struct
{
    unsigned __int16 Id1;
    unsigned __int32 Id2;
    unsigned __int32 Id;
    unsigned char length;
    bool rtr;
    bool st_ext;
    unsigned char data[8];
    CANMessageTime time;
} CAN_MESSAGE;
```

Id1	standardní část identifikátoru (11 bitů)
Id2	rozšířená část identifikátoru (18 bitů)
Id	identifikátor v 29 bitovém formátu
length	počet datových bytů zprávy
rtr	příznak zprávy RTR (Remote Transfer Request)
st_ext	rozlišení zda se jedná o zprávu se standardním nebo rozšířeným identifikátorem
data	pole datových bytů
time	struktura s časovou značkou

Tato struktura je definována v hlavičkovém souboru canbus.h. Prvek time obsahuje časovou značku, struktura CANMessageTime je definována ve stejném souboru.

Struktura CAN_MESSAGE obsahuje dva tvary identifikátoru, je to identifikátor ve tvaru 11+ 18 bitů a také 29 bitový identifikátor. Pro synchronizaci hodnot identifikátorů slouží tyto dvě funkce:

```
void CANMsgUpdateFrom11_18(CAN_MESSAGE *message);
void CANMsgUpdateFrom29(CAN_MESSAGE *message);
```

První updatuje 29 identifikátor z formátu 11+18, druhá pracuje opačně. API interně pracuje s formátem identifikátoru 11+18. V případě používání 29 bitového tvaru je nutno zabezpečit správnost

ID před odesláním zprávy na CAN pomocí funkce CANMsgUpdateFrom29. Při příjmu API zabezpečuje konverzi ID automaticky.

V souvislosti s definicí struktury CANovské zprávy jsou v tomto souboru definovány hlavičky pomocných funkcí, které slouží ke konverzi identifikátoru ze struktury CAN_MESSAGE na tvar uložený v registrech často používaných obvodů SJA1000. MCP2510, MCP2515, I82527, CC750 a CC770. Parametry těchto funkcí jsou dva, prvním je struktura CAN_MESSAGE. V ní je třeba pro konverzi vyplnit položky ID1, Id2, length, rtr a st_ext. Druhým parametrem je ukazatel na pole prvků unsigned char o délce 5. Pro každý obvod existují 2 varianty TX a RX pro tvar v transmit (TX) a receive (RX) bufferu.

```
void CANMsg2MCP251x_TX(CAN_MESSAGE message, unsigned char *data);
void CANMsg2MCP251x_RX(CAN_MESSAGE message, unsigned char *data);

void CANMsg2SJA1000_TX(CAN_MESSAGE message, unsigned char *data);
void CANMsg2SJA1000_RX(CAN_MESSAGE message, unsigned char *data);

void CANMsg2I82527_TX  (CAN_MESSAGE message, unsigned char *data);
void CANMsg2I82527_RX  (CAN_MESSAGE message, unsigned char *data);

void CANMsg2CC7x0_TX   (CAN_MESSAGE message, unsigned char *data);
void CANMsg2CC7x0_RX   (CAN_MESSAGE message, unsigned char *data);
```

Pro práci s některými high-level protokoly existuje několik funkcí pro vytváření identifikátoru:

```
void CANMsgSAE(CAN_MESSAGE *message
                  ,unsigned char Priority
                  ,bool DataPage
                  ,unsigned char PDUFormat
                  ,unsigned char DestinationAddress
                  ,unsigned char SourceAddress);

void CANMsgDeviceNet(CAN_MESSAGE *Message
                  ,unsigned char Group
                  ,unsigned char Data1
                  ,unsigned char Data2);

void CANMsgSDSShort(CAN_MESSAGE *message
                  ,bool Dir_Pri
                  ,unsigned char LogicalAddress
                  ,unsigned char ServiceType);


void CANMsgSDSLong(CAN_MESSAGE *Message
                  ,bool Dir_Pri
                  ,unsigned char LogicalAddress
                  ,unsigned char ServiceType
                  ,unsigned char Dlc
                  ,unsigned char ServiceSpecifiers
                  ,unsigned char EmbeddedObjectID
                  ,unsigned char ServiceParameters);

void CANMsgSDSFragmentedLong(CAN_MESSAGE *Message
                  ,bool Dir_Pri
                  ,unsigned char LogicalAddress
                  ,unsigned char ServiceType
                  ,unsigned char Dlc
                  ,unsigned char ServiceSpecifiers
                  ,unsigned char EmbeddedObjectID
                  ,unsigned char ServiceParameters
                  ,unsigned char FragmentNumber
                  ,unsigned char TotalFragmentBytes);
```

Dále je v tomto souboru definován výčtový typ CAN_SPEED. Jak je zřejmé, je tento typ určen pro zadání/uložení komunikační rychlosti. Jeho definice vypadá takto:

```
enum CAN_SPEED
{
    SPEED_10k   = 10,
    SPEED_20k   = 20,
    SPEED_33_3k = 33,
    SPEED_50k   = 50,
    SPEED_62_5k = 62,
    SPEED_83_3k = 83,
    SPEED_100k  = 100,
    SPEED_125k  = 125,
    SPEED_250k  = 250,
    SPEED_500k  = 500,
    SPEED_1M    = 1000,
    SPEED_USR   = 0,
};
```

Pro nastavení registrů sloužících k nastavení komunikační rychlosti a bodu vzorkování slouží struktury PP2CAN_timing a USB2CAN_timing. Jejich definice má tento tvar:

```
typedef struct
{
    unsigned char speed_10k[3];
    unsigned char speed_20k[3];
    unsigned char speed_33k[3];
    unsigned char speed_50k[3];
    unsigned char speed_62k[3];
    unsigned char speed_83k[3];
    unsigned char speed_100k[3];
    unsigned char speed_125k[3];
    unsigned char speed_250k[3];
    unsigned char speed_500k[3];
    unsigned char speed_1000k[3];
}PP2CAN_timing;

typedef struct
{
    unsigned char speed_10k[2];
    unsigned char speed_20k[2];
    unsigned char speed_33k[2];
    unsigned char speed_50k[2];
    unsigned char speed_62k[2];
    unsigned char speed_83k[2];
    unsigned char speed_100k[2];
    unsigned char speed_125k[2];
    unsigned char speed_250k[2];
    unsigned char speed_500k[2];
    unsigned char speed_800k[2];
    unsigned char speed_1000k[2];
}USB2CAN_timing;
```

Dále jsou definovány 2 globální proměnné PP2CAN_default_timing a USB2CAN_default_timing odvozené od uvedených struktur, jejichž hodnoty jsou přednastaveny na defaultní hodnoty a které jsou použity při inicializaci CAN interface. Hodnoty v těchto globálních proměnných je možno uživatelem modifikovat. Defaultní hodnoty jsou definovány takto:

```
#define MCP_10KB_CNF1_Osc20 0x27
#define MCP_10KB_CNF2_Osc20 0xBF
#define MCP_10KB_CNF3_Osc20 0x07

#define MCP_20KB_CNF1_Osc20 0x31
#define MCP_20KB_CNF2_Osc20 0xA0
#define MCP_20KB_CNF3_Osc20 0x02

#define MCP_33KB_CNF1_Osc20 0x1D
#define MCP_33KB_CNF2_Osc20 0xA0
#define MCP_33KB_CNF3_Osc20 0x02

#define MCP_50KB_CNF1_Osc20 0x13
#define MCP_50KB_CNF2_Osc20 0xA0
#define MCP_50KB_CNF3_Osc20 0x02

#define MCP_62KB_CNF1_Osc20 0x0F
#define MCP_62KB_CNF2_Osc20 0xA0
#define MCP_62KB_CNF3_Osc20 0x02

#define MCP_83KB_CNF1_Osc20 0x0B
#define MCP_83KB_CNF2_Osc20 0xA0
#define MCP_83KB_CNF3_Osc20 0x02

#define MCP_100KB_CNF1_Osc20 0x09
#define MCP_100KB_CNF2_Osc20 0xA0
#define MCP_100KB_CNF3_Osc20 0x02

#define MCP_125KB_CNF1_Osc20 0x07
#define MCP_125KB_CNF2_Osc20 0xA0
#define MCP_125KB_CNF3_Osc20 0x02

#define MCP_250KB_CNF1_Osc20 0x02
#define MCP_250KB_CNF2_Osc20 0xA0
#define MCP_250KB_CNF3_Osc20 0x02

#define MCP_500KB_CNF1_Osc20 0x01
#define MCP_500KB_CNF2_Osc20 0xA0
#define MCP_500KB_CNF3_Osc20 0x02

#define MCP_1000KB_CNF1_Osc20 0x00
#define MCP_1000KB_CNF2_Osc20 0xA0
#define MCP_1000KB_CNF3_Osc20 0x02
```

```
#define SJA_10KB_BTR0_Osc16 0x31
#define SJA_10KB_BTR1_Osc16 0xBA

#define SJA_20KB_BTR0_Osc16 0x18
#define SJA_20KB_BTR1_Osc16 0xBA

#define SJA_33KB_BTR0_Osc16 0x13
#define SJA_33KB_BTR1_Osc16 0xA7

#define SJA_50KB_BTR0_Osc16 0x09
#define SJA_50KB_BTR1_Osc16 0x1C

#define SJA_62KB_BTR0_Osc16 0x07
#define SJA_62KB_BTR1_Osc16 0xBA

#define SJA_83KB_BTR0_Osc16 0x07
#define SJA_83KB_BTR1_Osc16 0xA7

#define SJA_100KB_BTR0_Osc16 0x04
#define SJA_100KB_BTR1_Osc16 0x1C

#define SJA_125KB_BTR0_Osc16 0x03
#define SJA_125KB_BTR1_Osc16 0x1C

#define SJA_250KB_BTR0_Osc16 0x01
#define SJA_250KB_BTR1_Osc16 0x1C

#define SJA_500KB_BTR0_Osc16 0x00
#define SJA_500KB_BTR1_Osc16 0x1C

#define SJA_800KB_BTR0_Osc16 0x00
#define SJA_800KB_BTR1_Osc16 0x16

#define SJA_1000KB_BTR0_Osc16 0x00
#define SJA_1000KB_BTR1_Osc16 0x14
```

Popis obsahu souboru canport.h lze ukončit uvedením několika pomocných funkcí. Prvních 6 funkcí slouží k převodu komunikační rychlostí ve tvaru CAN_SPEED, KBaud a Int. Ty lze použít například pro konfigurační soubory nebo dialogy.

```
CAN_SPEED KBaud2CANSpeed(int speed);
CAN_SPEED Int2CANSpeed(int speed);

int CANSpeed2Int(CAN_SPEED speed);
int CANSpeed2KBaud(CAN_SPEED speed);

int KBaud2Int(int speed);
int Int2KBaud(int speed);
```

Funkce provádějí konverzi dle následující tabulky:

CAN_SPEED	Int	KBaud
SPEED_10k	0	10
SPEED_20k	1	20
SPEED_33_3k	2	33
SPEED_50k	3	50
SPEED_62.5k	4	62
SPEED_83_3k	5	83
SPEED_100k	6	100
SPEED_125k	7	125
SPEED_250k	8	250
SPEED_500k	9	500
SPEED_1000k	10	1000

```
void Byte2BinString(unsigned char data, char *text);
```

Tato poslední funkce převádí číslo typu unsigned char na řetězec délky 8 v binárním formátu. znamená např. 3 dekadicky na řetězec 00000011.

Soubor bitbase.h obsahuje následující makra pro testování stavu bitů:

```
#define hw_bitset(var,bitno) ((var)|=1<<(bitno))
#define hw_bitclr(var,bitno) ((var)&=~(1<<(bitno)))
#define hw_bitest(var,bitno) (((var)>>(bitno))&0x01)
#define hw_bittest(var,bitno) (hw_bitest(var,bitno))
```

Soubory MCP2510 a SJA1000 obsahují definice adres registrů a bitových masek pro CAN bus řadiče SJA1000, MCP2510 a MCP2515.

Nyní se dostáváme od pomocných funkcí a definic k popisu vlastního X2CAN API. Nejprve začneme popisem funkcí pro adaptér PP2CAN, následovat bude popis funkcí pro USB2CAN a virtuální port V2CAN a zakončíme popisem zastřešení prostřednictvím rozhraní X2CAN. Funkce pro adaptéry PP2CAN a USB2CAN lze použít samostatně, bez nutnosti zastřešení X2CAN, nicméně X2CAN dovoluje v hotové aplikaci jednoduchou záměnu obou adapterů a tak je jeho použití doporučeno. Nicméně znalost API PP2CAN a USB2CAN vám dovolí použít "specialit" těchto převodníků a dovolí hlouběji pochopit princip jejich funkce.

3. Funkce API PP2CAN

Existují 4 varianty tohoto adaptéru, (High speed revize 0 a 1, Low speed revize 0 a Single wire revize 0), proto jsou v souboru PP2CAN.h definovány tyto konstanty potřebné při inicializaci adapteru:

```
#define PP2CAN_HW_HIGH_SPEED_0    0
#define PP2CAN_HW_HIGH_SPEED_1    1
#define PP2CAN_HW_LOW_SPEED_0     2
#define PP2CAN_HW_SINGLE_WIRE_0   3
```

Samotná inicializace se provádí voláním funkce PP2CAN_Open. Ne jednom PC může být současně provozován pouze 1 adaptér PP2CAN. Adaptérů USB2CAN může být paralelně provozováno libovolné množství.

```
bool PP2CAN_Open(WORD Address
    , CAN_SPEED Speed
    , void (*error_function)(int err_code, const char * error_string)
    , void (*msg_receiver)(MCP2510Msg *msg)
    , int HW_Version
    , bool OneShotMode
    , bool PassiveMode
    , int ThreadPriority );
```

Funkce má tyto parametry:

Address	Adresa paralelního portu. Tuto adresu lze zjistit nejlépe ve Vlastnosti systému -> Správce zařízení -> Porty (COM a LPT) -> Port tiskárny (LPTx) -> Prostředky - Rozsah I/O. Zde obvykle bývá jedna z hodnot: 0x378, 0x278, 0x3BC.
Speed	Komunikační rychlost.
error_function	Ukazatel na funkci, která je volána při výskytu chyby. Funkci je předán chybový kód a textový popis.
msg_receiver	Ukazatel na funkci, která je volána při příjmu zprávy. Funkce má jako parametr strukturu se zprávou. Pokud je tento parametr NULL, žádná funkce se nevolá, zpracování je prováděno přístupem uživatelské aplikace k bufferu přijatých zpráv. V praxi se doporučuje tento druhý způsob.
HW_Version	Verze HW PP2CAN (například nejčastěji PP2CAN_HW_HIGH_SPEED_1).
OneShotMode	Zpráva je odesílána pouze jednou, odeslání není při nepotvrzení opakováno.
PasssiveMode	Parametr nastavuje adapter do stavu, kdy není možno odeslat zprávu, je možný pouze příjem. Zabraňuje nechtěnému odeslání zprávy, zvyšuje množství zachycených zpráv.
ThreadPriority	Priorita vlákna pro komunikaci s adapterem.. THREAD_PRIORITY_TIME_CRITICAL=0, THREAD_PRIORITY_HIGHEST=1, THREAD_PRIORITY_NORMAL=2.

Funkce PP2CAN_Open provede inicializaci s defaultním nastavením filtrů kdy jsou přijímány všechny zprávy a uvede adaptér do pracovního režimu. Vrací true pokud inicializace proběhla v pořádku. Případné chyby jsou signalizovány voláním chybové funkce (error_function) a vrácením hodnoty false.

Pro uzavření portu (ukončení komunikace) se používá funkce PP2CAN_Close. Funkce nemá žádné parametry.

Zařízení má 4 pracovní režimy. Tyto režimy jsou definovány ve výčtovém typu PP2CAN_MODE. Zde je jeho definice:

```
enum PP2CAN_MODE
{
    PP2CAN_MODE_CONFIG = 0,
    PP2CAN_MODE_NORMAL,
    PP2CAN_MODE_LOOPBACK,
    PP2CAN_MODE_LISTEN_ONLY,
};
```

Pro nastavení a zjištění pracovního režimu jsou určeny funkce:

```
void PP2CAN_SetMode(PP2CAN_MODE mode);
void PP2CAN_SetConfigMode(void);
```

```
void PP2CAN_SetNormalMode(void);
void PP2CAN_SetLoopbackMode(void);
void PP2CAN_SetListenOnlyMode(void);
PP2CAN_MODE PP2CAN_GetMode(void);
```

V režimu PP2CAN_MODE_CONFIG lze provádět změnu komunikační rychlosti, nastavení filtrů zpráv, režimu OneShotMode a podobně. V režimu PP2CAN_MODE_NORMAL může adaptér přijímat a odesílat zprávy. V režimu PP2CAN_MODE_LOOPBACK jsou odeslané zprávy přijaty zpět prostřednictvím HW loopbacku. Režim PP2CAN_MODE_LISTEN_ONLY dovoluje pouze přijímat zprávy, jsou však přijaty i zprávy poškozené.

Pokud volání PP2CAN_Open proběhlo v pořádku, jsou přijímány zprávy a je možno zprávy na sběrnici odesílat. K odeslání zpráv na sběrnici CAN jsou určeny tyto funkce:

```
bool PP2CAN_SendMessage(MCP2510Msg &data);
bool PP2CAN_SendCANMessage(CAN_MESSAGE message);
bool PP2CAN_SendRegisterMessage(unsigned char Data[13]);
bool PP2CAN_SendMCPMessage(MCP2510Msg *Data);
bool PP2CAN_SendStandardMessage(unsigned __int16 StandardId
        , bool RTR
        , unsigned char Length
        , unsigned char *Data);
bool PP2CAN_SendExtendedMessage(unsigned __int16 StandardId
        , unsigned __int32 ExtendedId
        , bool RTR
        , unsigned char Length
        , unsigned char *Data);
bool PP2CAN_SendExtendedMessage29(unsigned __int32 ExtendedId
        , bool RTR
        , unsigned char Length
        , unsigned char *Data);
```

Popis struktury CAN_MESSAGE již byl uveden. Funkce se strukturou MCP2510Msg jsou implementovány pro zpětnou kompatibilitu s PP2CAN API v1.x. Tato struktura je definována takto:

```
typedef union
{
    unsigned char bytes[13];
    struct
    {
        unsigned char ID[4];
        unsigned char DLC;
        unsigned char data[8];
    } item;
} MCP2510MsgData;

typedef struct
{
    MCP2510MsgData msg;
    CANMessageTime time;
} MCP2510Msg;
```

Práci se strukturou MCP2510MsgData usnadňuje několik funkcí pro získání a nastavení identifikátoru, typu zprávy (standardní/rozšířený identifikátor, data/rtr frame). Jako parametr data se předává adresa prvního bajtu struktury.

```
bool PP2CAN_GetId1(unsigned char *data
                ,unsigned __int16 &Id1
                ,unsigned __int32 &Id2);

bool PP2CAN_GetId2(unsigned char *data
                ,unsigned __int16 *Id1
```

```
                ,unsigned __int32 *Id2);

void PP2CAN_SetExtendedId(unsigned char *data
                ,unsigned __int16 Id1
                ,unsigned __int32 Id2);

void PP2CAN_SetStandardId(unsigned char *data, unsigned __int16 Id1);
void PP2CAN_SetExtendedId29(unsigned char *data, unsigned __int32 Id);

bool PP2CAN_SetDLC(unsigned char *data, bool RTR, unsigned char length);
void PP2CAN_GetDLC1(unsigned char *data, bool &RTR, unsigned char &length);
void PP2CAN_GetDLC2(unsigned char *data, bool *RTR, unsigned char *length);
```

Pro čtení přijatých zpráv ze vstupního bufferu je možno použít tyto funkce:

```
bool PP2CAN_GetMessage(MCP2510Msg **data);
bool PP2CAN_GetMCPMessage(MCP2510Msg *data);
bool PP2CAN_GetCANMessage(CAN_MESSAGE *message);
bool PP2CAN_GetMessage11_18(bool *StExt
            , unsigned __int16 *StandardId
            , unsigned __int32 *ExtendedId
            , bool *RTR
            , unsigned char *Length
            , unsigned char *Data);
bool PP2CAN_GetMessage29(bool *StExt
            , unsigned __int32 *Id
            , bool *RTR
            , unsigned char *Length
            , unsigned char *Data);
void PP2CAN_DeleteMessage(MCP2510Msg *data);
```

Je-li zpráva vyčtena prostřednictvím funkce PP2CAN_GetMessage, je třeba tuto zprávu po zpracování odstranit z paměti voláním funkce PP2CAN_DeleteMessage. V případě, že buffer neobsahuje žádné zprávy, vracejí uvedené funkce false.

Pro zjištění počtu zpráv v bufferu pro odeslání je určena funkce:

```
int PP2CAN_GetTXBufferLength(void);
```

Analogicky pro počet přijatých zpráv, které čekají na zpracování je určena funkce:

```
int PP2CAN_GetRXBufferLength(void);
```

Buffery je možno vyprázdnit voláním:

```
void PP2CAN_ClearBuffers(void);
```

Pro čekání na příchod zprávy například ve vláknu je pak možno použít funkci:

```
bool USB2CAN_WaitForRxMessage(unsigned int Timeout);
```

Tato funkce vrací true, pokud byla přijata zpráva, v případě že v časovém intervalu, který je zadán jako parametr timeout, není žádná zpráva obdržena, vrací funkce false. Nekonečné čekání na příchod zprávy je možné realizovat zadáním hodnoty INFINITE.

Z uživatelského hlediska jsou dále zajímavé funkce pro monitorování hodnot registrů TEC (Transmit Error Counter), REC (Receive Error Counter) a čítače RST (Reset Counter). Význam registrů TEC a REC je blíže vysvětlen ve specifikaci sběrnice CAN. Čítač RST počítá restarty adaptéru při přechodu do stavu Bus-off. U jednoduchého adaptéru PP2CAN ovlivňuje čtení registrů TEC a REC množství odeslaných a zachycených zpráv. Pro velké množství přenášených zpráv je

vhodné čtení těchto registrů zakázat. K povolení/zakázání slouží následující 2 funkce. Další 3 funkce jsou určeny pro vlastní čtení uvedených registrů (čítače).

```
void PP2CAN_EnableReadTEC(bool enable);
void PP2CAN_EnableReadREC(bool enable);

unsigned char PP2CAN_GetREC(void);
unsigned char PP2CAN_GetTEC(void);
unsigned int PP2CAN_GetRST(void);
```

Vlastní PP2CAN obsahuje velké množství dalších funkcí, můžeme například zcela vyřadit vlákno pro komunikaci s adaptérem a nahradit jej vlastním s využitím funkcí API pro zápis, čtení a modifikaci registrů obvodu MCP251x, provádět nastavení filtrů zpráv atd. Popis těchto dalších funkcí lze nalézt v hlavičkovém souboru pp2can.h.

4. Funkce API USB2CAN

Narozdíl od adaptéru PP2CAN, který může být v PC aktivní v jeden okamžik pouze jeden, je možno na PC provozovat v jeden okamžik prakticky libovolné množství CAN bus adaptérů USB2CAN paralelně. Proto jsou pro výběr a prací s jednotlivými adaptéry definovány funkce:

```
void* USB2CAN_PrepareAdapter(void);
void* USB2CAN_PrepareAdapterEx(EUSB2CANDevice selector, char *name);
void USB2CAN_SelectActualAdapter(void* adapter);
void USB2CAN_DestroyAdapter(void* adapter);
```

První funkce USB2CAN_PrepareAdapter vrací ukazatel typu void na řídící datovou strukturu, ta je nutná pro práci adaptéru, který se snažíme otevřít. Funkce USB2CAN_PrepareAdapter otevře první nalezený adaptér, pokud chceme adapter přesněji specifikovat, použijeme funkci USB2CAN_PrepareAdapterEx. Tato funkce má 2 parametry, pomocí prvního specifikujeme metodu výběru adaptéru. Adaptér lze vybrat podle jména (to je defaultně stejné pro všechny, lze změnit pomocí utility FTDI, sériového čísla a čísla připojení adaptéru). V praxi pak jednotlivé varianty vypadají takto:

```
enum EUSB2CANDevice
{
    OPEN_BY_DEVICE_NUMBER = 0,
    OPEN_BY_SERIAL_NUMBER = FT_OPEN_BY_SERIAL_NUMBER,    // 1
    OPEN_BY_DESCRIPTION   = FT_OPEN_BY_DESCRIPTION,      // 2
};

void *my_can_adapter;

// Prvni pripojene zarizeni
my_can_adapter = USB2CAN_PrepareAdapterEx(OPEN_BY_DEVICE_NUMBER, "0");

// Druhé pripojene zarizeni
my_can_adapter = USB2CAN_PrepareAdapterEx(OPEN_BY_DEVICE_NUMBER, "1");

// Seriove cislo
my_can_adapter = USB2CAN_PrepareAdapterEx(FT_OPEN_BY_SERIAL_NUMBER, "000112");

// Adapter USB2CAN s konfiguracni pameti
my_can_adapter = USB2CAN_PrepareAdapterEx(FT_OPEN_BY_DESCRIPTION, "USB2CAN");

// Adapter USB2CAN bez konfiguracni pameti
my_can_adapter = USB2CAN_PrepareAdapterEx(FT_OPEN_BY_DESCRIPTION
                                        , "USB <-> Serial");
```

Výběr adaptéru se kterým chceme v daný okamžik pracovat, provádíme pomocí funkce USB2CAN_SelectActualAdapter. Po uzavření a ukončení práce s tímto adaptérem pak data odstraníme z paměti voláním USB2CAN_DestroyAdapter.

Pro otevření a uzavření vlastního CAN portu adapteru používáme funkci:

```
bool USB2CAN_Open(CAN_SPEED speed
        , bool CreateCommThreads
        , void (*error_function)(int err_code, const char * error_string)
        , bool LowSpeed);
bool USB2CAN_Close(void);
```

Prvním parametrem funkce USB2CAN_Open je komunikační rychlost. Druhý parametr nastavujeme na true, pouze v případě, kdy používáme vlastní mechanismy pro komunikaci se zařízením prostřednictvím USB, je tento parametr nastaven na false. Třetím parametrem je ukazatel na chybovou funkci, tento ukazatel a jeho funkce je stejná jako u adaptéru PP2CAN. Poslední parametr udává zda budeme pracovat s high nebo low speed variantou. Pro low-speed má tento parametr hodnotu true. V případě úspěšné inicializace adaptéru a otevření CAN portu, vrací tato funkce true. Pro uzavření portu používáme funkcí USB2CAN_Close.

Po úspěšném otevření portu již máme k dispozici obvyklou skupinu funkcí pro příjem a odeslání zpráv prostřednictvím CAN sběrnice:

```
bool USB2CAN_SendMessage(SJA1000MsgData &data);

bool USB2CAN_SendCANMessage(CAN_MESSAGE message);

bool USB2CAN_SendRegisterMessage(unsigned char Data[13]);

bool USB2CAN_SendSJAMessage(SJA1000MsgData *Data);

bool USB2CAN_SendStandardMessage(unsigned __int16 StandardId
                         ,bool RTR
                         ,unsigned char Length
                         ,unsigned char *Data);

bool USB2CAN_SendExtendedMessage(unsigned __int16 StandardId¨
                         ,unsigned __int32 ExtendedId
                         ,bool RTR
                         ,unsigned char Length
                         ,unsigned char *Data);

bool USB2CAN_SendExtendedMessage29(unsigned __int32 ExtendedId
                         ,bool RTR
                         ,unsigned char Length
                         ,unsigned char *Data);

bool USB2CAN_GetMessage(SJA1000MsgData **data); - ZRUŠENO

bool USB2CAN_GetSJAMessage(SJA1000MsgData *data);

bool USB2CAN_GetCANMessage(CAN_MESSAGE *message);

bool USB2CAN_GetMessage11_18(bool *StExt
                         ,unsigned __int16 *StandardId
                         ,unsigned __int32 *ExtendedId
                         ,bool *RTR
                         ,unsigned char *Length
                         ,unsigned char *Data);
```

```
bool USB2CAN_GetMessage11_18(bool* StExt
                            ,unsigned __int16* StandardId
                            ,unsigned __int32* ExtendedId
                            ,bool* RTR
                            ,unsigned char* Length
                            ,unsigned char* Data
                            ,CANMessageTime* time);

bool USB2CAN_GetMessage29(bool *StExt
                            ,unsigned __int32 *Id
                            ,bool *RTR
                            ,unsigned char *Length
                            ,unsigned char *Data);
```

Funkce USB2CAN_GetSJAMessage a USB2CAN_GetCANMessage očekávají platný ukazatel. Na toto místo jsou pak zapsána data zprávy.

```
void USB2CAN_DeleteMessage(SJA1000MsgData *data);  - ZRUŠENO
```

Zprávu přijatou pomocí USB2CAN_GetMessage nemažeme pomocí operátoru delete ale touto funkcí.

```
bool USB2CAN_WaitForRxMessage(unsigned int timeout);
```

Čekání na příchod CAN zprávy.

```
int USB2CAN_GetTXBufferLength();
```

Vrací velikost bufferu přijatých zpráv.

```
int USB2CAN_GetRXBufferLength();
```

Vrací velikost bufferu zpráv které čekají na odeslání.

```
void USB2CAN_SetTimeStampMode (bool mode);
```

Povolení přesného měření času příjmu zprávy.

```
void USB2CAN_ClearBuffers (void);
```

Vyprázdní buffery pro příjem a odeslání CANovských zpráv.

```
unsigned char USB2CAN_GetREC (void);
```

Vrací hodnotu Receive Error Counteru.

```
unsigned char USB2CAN_GetTEC (void);
```

Vrací hodnotu Transmit Error Counteru.

```
unsigned int USB2CAN_GetRST (void);
```

Vrací hodnotu Reset Counteru (počet resetů SJA1000).

```
void USB2CAN_EnableReadTEC (bool enable);
```

Povolení automatického čtení TEC.

```
void USB2CAN_EnableReadREC (bool enable);
```

Povolení automatického čtení TEC.

Uvedené funkce jsou analogické funkcím pro adaptér PP2CAN, za zmínku stojí struktura SJA1000MsgData. Ta má tento tvar:

```
typedef union
{
    unsigned char bytes[13];
    struct {
        unsigned char DLC;
        unsigned char ID[2];
        unsigned char data[8];
    } standard;
    struct {
        unsigned char DLC;
        unsigned char ID[4];
        unsigned char data[8];
    } extended;
} SJA1000MsgData;
```

Pro případ nízko-úrovňového řízení (pro majitele vývojové dokumentace) je připravena sada následujících funkcí:

```
void USB2CAN_SetupBasic(int BaudRate);
```

Defaultni inicializace SJA1000 a USB2CANu, odpovídá inicializaci která je provedena pro USB2CAN_Open.

```
void USB2CAN_SetTimeout(int ms);
```

Funkce pro nastavení timeoutu pro odpovědi (potvrzení) adaptéru na zprávy zaslané z PC.

```
bool USB2CAN_Loopback();
```

Odeslání zprávy USB_LOOPBACK, je-li přijata zpět odpověď adaptéru, vrací true.

```
bool USB2CAN_SetMode(int Mode);
```

Nastavení pracovního módu (BOOT, CONFIG, NORMAL, LOOPBACK).

```
bool USB2CAN_GetMode(int *Mode);
```

Funkce pro zjištění aktuálního pracovního módu. Pokud adaptér nezašle aktuální mód, do vypršení timeoutu, vrací false.

```
bool USB2CAN_GetFirmwareVersion(char *Version);
```

Zjištění aktuální verze firmware. Pokud adaptér nezašle odpověď do vypršení timeoutu, vrací false.

```
bool USB2CAN_Command0(unsigned char Command);

bool USB2CAN_Command1(unsigned char Command
                ,unsigned char Param1);

bool USB2CAN_Command2(unsigned char Command
                ,unsigned char Param1
                ,unsigned char Param2);
```

```
bool USB2CAN_Command3(unsigned char Command
                ,unsigned char Param1
                ,unsigned char Param2
                ,unsigned char Param3);
```

Funkce jsou určeny pro zasílání speciálních příkazů skupiny COMMAND (viz. vývojová dokumentace).

```
bool USB2CAN_ReadReg(unsigned char Address, unsigned char *Data);
```

Funkce je určena pro čtení hodnoty specifikovaného registru obvodu SJA1000 v adaptéru USB2CAN.

```
bool USB2CAN_WriteReg(unsigned char Address, unsigned char Data);
```

Funkce je určena pro zápis hodnoty do specifikovaného registru obvodu SJA1000 v adaptéru USB2CAN.

```
bool USB2CAN_WriteReadReg(unsigned char Address´
                ,unsigned char Data
                ,unsigned char *DataOut);
```

Funkce je určena pro čtení a zpětné čtení hodnoty specifikovaného registru obvodu SJA1000 v adaptéru USB2CAN.

```
bool USB2CAN_BitModReg(unsigned char Address
                ,unsigned char Mask
                ,unsigned char Data);
```

Bitová modifikace pomocí hodnoty a masky registru obvodu SJA1000.

```
bool USB2CAN_BitModReadReg(unsigned char Address
                ,unsigned char Mask
                ,unsigned char Data
                ,unsigned char *DataOut);
```

Bitová modifikace pomocí hodnoty a masky a zpětné čtení hodnoty registru obvodu SJA1000.

```
bool USB2CAN_WriteInstruction(unsigned __int16 Address
                ,unsigned __int16 Instruction);
```

Zápis instrukce do programové paměti (změna firmware) řídícího mikroprocesoru.

```
bool USB2CAN_ReadTEC(unsigned char *TEC);
```

Příkaz zašle dotaz na stav registru TEC (Transmit Error Counter), zpět je vrácena hodnota tohoto registru. Hodnota je uložena na adresu specifikovanou v TEC. Není li obdržena odpověď do vypršení timeoutu, je vráceno false. Doporučena varianta je vsak povoleni automatického čteni pomoci USB2CAN_EnableReadTEC a cteni hodnoty pomoci USB2CAN_GetTEC.

```
bool USB2CAN_ReadREC(unsigned char *REC);
```

Příkaz zašle dotaz na stav registru REC (Receive Error Counter), zpět je vrácena hodnota tohoto registru. Není li obdržena odpověď do vypršení timeoutu, je vráceno false.

```
bool USB2CAN_ReadRST(unsigned int *RST);
```

Příkaz zašle dotaz na stav čítače RST (Reset Error Counter), zpět je vrácena hodnota tohoto registru. Není li obdržena odpověď do vypršení timeoutu, je vráceno false. Tento čítač udává počet přechodů do stavu Bus-Off (a restartů SJA1000).

```
void USB2CAN_CmdPeliCAN(void);
```

Nastavení pracovního režimu PeliCAN obvodu SJA1000.

```
void USB2CAN_CmdResetMode(void);
```

Nastavení konfiguračního režimu obvodu SJA1000.

```
void USB2CAN_CmdOperatingMode(void);
```

Nastavení základního pracovního režimu obvodu SJA1000.

```
void USB2CAN_CmdBaudRate(unsigned char t0, unsigned char t1);
```

Zápis hodnot do timming registrů obvodu SJA1000. Možno pouze v konfiguračním režimu.

```
void USB2CAN_CmdEnableReadTEC(bool Enable);
```

Povolení čtení registru TEC obvodu SJA1000.

```
void USB2CAN_CmdEnableReadREC(bool Enable);
```

Povolení čtení registru REC obvodu SJA1000.

5. Funkce API V2CAN

Začlenění virtuálního CAN bus portu dovoluje v aplikacích, které jsou postaveny na X2CAN API fungovat i bez připojeného CAN bus adaptéru. CAN zpráva odeslaná pomocí virtuálního portu je přijata zpět. To dovoluje například provádět off-line analýzu zalogovaných dat a zejména zjednodušuje testování vyvíjeného SW postaveného na X2CAN API. K dispozici je obvyklá sada funkcí, tentokrát s předponou V2CAN.

```
bool V2CAN_Open();
bool V2CAN_Close();

bool V2CAN_SendRegisterMessage(unsigned char Data[13]);

bool V2CAN_SendStandardMessage(unsigned __int16 StandardId
                ,bool RTR
                ,unsigned char Length
                ,unsigned char *Data);

bool V2CAN_SendExtendedMessage(unsigned __int16 StandardId
                ,unsigned __int32 ExtendedId
                ,bool RTR, unsigned char Length
                ,unsigned char *Data);

bool V2CAN_SendExtendedMessage29(unsigned __int32 ExtendedId
                ,bool RTR
                ,unsigned char Length
                ,unsigned char *Data);

bool V2CAN_SendCANMessage(CAN_MESSAGE message);

bool V2CAN_GetMessage11_18(bool *StExt
                ,unsigned __int16 *StandardId
                ,unsigned __int32 *ExtendedId
                ,bool *RTR
                ,unsigned char *Length
                ,unsigned char *Data);
```

```
bool V2CAN_GetMessage29(bool *StExt
                ,unsigned __int32 *Id
                ,bool *RTR, unsigned char *Length
                ,unsigned char *Data);

bool V2CAN_GetCANMessage(CAN_MESSAGE *message);

void V2CAN_SetTimeStampMode(bool mode);
int V2CAN_GetRXBufferLength(void);
int V2CAN_GetTXBufferLength(void);
void V2CAN_ClearBuffers(void);
bool V2CAN_WaitForRxMessage(unsigned int timeout);

unsigned char V2CAN_GetREC(void);
unsigned char V2CAN_GetTEC(void);
unsigned int V2CAN_GetRST(void);
void V2CAN_EnableReadTEC(bool enable);
void V2CAN_EnableReadREC(bool enable);
```

6. Funkce API X2CAN

Aby aplikace byly nezávislé na skutečně použitém CAN adaptéru, obsahuje X2CAN API skupinu funkcí, která nám toto dovoluje. Pouze při inicializaci (otevření) CAN portu je nutno specifikovat použitý adaptér. Další funkce pro odeslání a příjem zpráv jsou shodné pro PP2CAN, USB2CAN i V2CAN. V současné verzi je však podporována současná práce pouze více adaptérů USB2CAN. Pomocí X2CAN nelze paralelně pracovat s jedním adaptérem PP2CAN a jedním a více adaptéry USB2CAN. V případě potřeby této kombinace je možno pracovat s USB2CAN adaptéry pomocí X2CAN API nebo USB2CAN API a s adaptérem PP2CAN pak pomocí PP2CAN API.

```
void X2CAN_Prepare(void);
```

Inicializace datových struktur.

```
bool X2CAN_Open(CAN_INTERFACE interface_type
          ,CAN_SPEED speed
          ,void (*error_function)(int err_code, const char * error_string) );
```

Otevření CAN portu, je specifikován CAN adaptér, komunikační rychlost a chybová funkce.

```
bool X2CAN_Open_PP2CAN(WORD address
          ,CAN_SPEED speed
          ,void (*error_function)(int err_code, const char * error_string)
          ,int HW_Version
          ,bool OneShotMode
          ,bool PassiveMode
          ,int ThreadPriority );
```

Otevření CAN portu PP2CAN, je specifikován CAN adaptér, komunikační rychlost a chybová funkce. Funkce dovoluje nastavit módy OneShotMode, Passive a prioritu vláken.

```
bool X2CAN_Open_V2CAN(void);
```

Otevření V2CAN portu.

```
bool X2CAN_Open_USB2CAN(CAN_SPEED speed
          ,void (*error_function)(int err_code, const char * error_string)
          ,bool low_speed);
```

Otevření USB2CAN portu. Před použitím je třeba provést volání USB2CAN_PrepareAdapter a SelectActualAdapter.

```
bool X2CAN_Close();
```

Uzavření portu.

```
bool X2CAN_IsInitialized();
```

Vrací true, je li port inicializován.

```
bool X2CAN_IsPP2CAN();
```

Vrací true, je-li aktuálně používaný adaptér typu PP2CAN.

```
bool X2CAN_IsUSB2CAN();
```

Vrací true, je-li aktuálně používaný adaptér typu USB2CAN.

```
bool X2CAN_IsV2CAN();
```

Vrací true, je-li aktuálně používaný CAN port typu V2CAN.

```
CAN_INTERFACE X2CAN_GetInterfaceType();
```

Vrací typ aktuálně používaného CAN adaptéru.

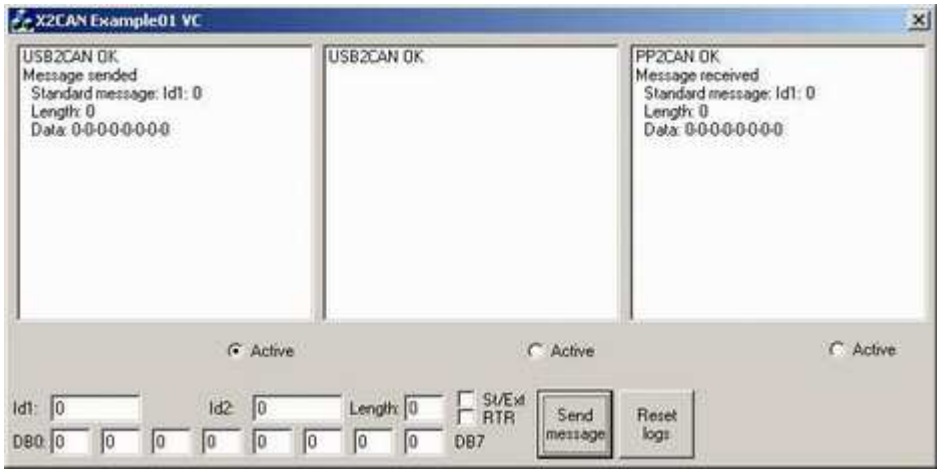
Dále pak máme k dispozici obvyklou skupinu funkcí pro odesílání/příjem zpráv, čtení velikosti bufferů a nejrůznějších příznaků. Tato skupina funkcí je ve skutečnosti realizována jako ukazatele na funkce jednotlivých adaptérů. Nemusíme se však omezovat jen na použití těchto funkcí, volání lze kombinovat se specializovanými funkcemi API jednotlivých adaptérů. Nicméně to je nutné jen ve výjimečných případech, v naprosté většině případů vystačíme s touto skupinou funkcí která dovoluje odesílat zprávy a číst zprávy z CAN sběrnice:

```
bool X2CAN_SendStandardMessage(unsigned __int16 StandardId
          ,bool RTR
          ,unsigned char Length
          ,unsigned char *Data);

bool X2CAN_SendExtendedMessage(unsigned __int16 StandardId
          ,unsigned __int32 ExtendedId
          ,bool RTR
          ,unsigned char Length
          ,unsigned char *Data);

bool X2CAN_SendExtendedMessage29(unsigned __int32 ExtendedId
          ,bool RTR
          ,unsigned char Length
          ,unsigned char *Data);

bool X2CAN_SendCANMessage(CAN_MESSAGE message);

bool X2CAN_GetMessage11_18(bool *StExt
          ,unsigned __int16 *StandardId
          ,unsigned __int32 *ExtendedId
          ,bool *RTR
          ,unsigned char *Length
          ,unsigned char *Data);

bool X2CAN_GetMessage29(bool *StExt
          ,unsigned __int32 *Id
          ,bool *RTR
          ,unsigned char *Length
          ,unsigned char *Data);
```

```
bool X2CAN_GetCANMessage(CAN_MESSAGE *message);

void X2CAN_SetTimeStampMode(bool mode);
int X2CAN_GetRXBufferLength(void);
int X2CAN_GetTXBufferLength(void);
void X2CAN_ClearBuffers(void);

bool X2CAN_WaitForRxMessage(unsigned int timeout);

unsigned char X2CAN_GetREC(void);
unsigned char X2CAN_GetTEC(void);
unsigned int X2CAN_GetRST(void);
void X2CAN_EnableReadTEC(bool enable);
void X2CAN_EnableReadREC(bool enable);
```

7. Example 01

Tento příklad je určen pro Microsoft Visual Studio C++. Byl vytvořen ve verzi 6, není však problém jej naimportovat i do verze NET. Tento příklad demonstruje elementární práci současně s dvěma USB2CAN adaptéry a jedním adaptérem PP2CAN. Pro jeho spuštění však není nutné mít připojené všechny. Na následujícím obrázku je vidět okno aplikace. Ta obsahuje pole pro vyplnění zprávy pro odeslání a 3 logovací okna pro jednotlivé CAN bus adaptéry. Komunikační rychlost je nastavena "natvrdo" v kódu. Všechny adaptéry jsou typu High speed. Kompletní zdrojový kód je ke stažení zde, do projektu je třeba zařadit knihovnu X2CAN API.



Jedná se o MFC aplikaci, typu Dialog based. Pro adaptéry je využito specifických API PP2CAN a USB2CAN. Třída hlavního dialogu je pojmenována na CExampe01_VCDlg. Inicializace adaptérů probíhá v metodě OnInitDialog, která je členem této třídy. O inicializaci se stará tato část kódu:

```
// Vytvorime datove struktury pro adaptery USB2CAN
// Otevirame adaptery parametru device number
// Prvni adapter
USB2CAN_adapter_1 = USB2CAN_PrepareAdapterEx(OPEN_BY_DEVICE_NUMBER, "0");
// Druhy adapter
USB2CAN_adapter_2 = USB2CAN_PrepareAdapterEx(OPEN_BY_DEVICE_NUMBER, "1");

// Nastaveni se kterym adapterem budu pracovat
USB2CAN_SelectActualAdapter(USB2CAN_adapter_1);
// Otevreni adapteru
bool back = USB2CAN_Open(SPEED_125k, TRUE ,ErrorUSB2CAN_1, false);
if(!back) m_log_usb2can_1.AddString("USB2CAN Error");
else m_log_usb2can_1.AddString("USB2CAN OK");
```

```
// Nastaveni se kterym adapterem budu pracovat
USB2CAN_SelectActualAdapter(USB2CAN_adapter_2);
// Otevreni adapteru
back = USB2CAN_Open(SPEED_125k, TRUE ,ErrorUSB2CAN_2, false);
if(!back) m_log_usb2can_2.AddString("USB2CAN Error");
else m_log_usb2can_2.AddString("USB2CAN OK");

// Otevreni adapteru PP2CAN, adresa paralelniho portu je 888 dekadicky (378h)
back = PP2CAN_Open(888,SPEED_125k,
ErrorPP2CAN,NULL,PP2CAN_HW_HIGH_SPEED_1,false,false,1);
if(!back) m_log_pp2can.AddString("PP2CAN Error");
else m_log_pp2can.AddString("PP2CAN OK");
```

Při ukončení aplikace probíhá uzavření portů:

```
// Ukonceni prace s adaptery
// Vyber aktualniho adapteru USB2CAN
USB2CAN_SelectActualAdapter(USB2CAN_adapter_1);
// Uzavreni CAN portu
USB2CAN_Close();
USB2CAN_DestroyAdapter(USB2CAN_adapter_1);

// Vyber aktualniho adapteru USB2CAN
USB2CAN_SelectActualAdapter(USB2CAN_adapter_2);
// Uzavreni CAN portu
USB2CAN_Close();
USB2CAN_DestroyAdapter(USB2CAN_adapter_2);

// Uzavreni adapteru PP2CAN
PP2CAN_Close();
```

Odeslání zprávy při stisku tlačítka ve funkci OnSend:

```
// CAN zprava kterou chceme odeslat
CAN_MESSAGE message;
UpdateData(TRUE);
// Vyplneni datovych polozek zpravy
message.Id1 = m_id1;
message.Id2 = m_id2;
(m_st_ext) ? message.st_ext = true : message.st_ext = false;
(m_rtr) ? message.rtr = true : message.rtr = false;
message.length = m_length;

message.data[0] = m_db0;
message.data[1] = m_db1;
message.data[2] = m_db2;
message.data[3] = m_db3;
message.data[4] = m_db4;
message.data[5] = m_db5;
message.data[6] = m_db6;
message.data[7] = m_db7;

// Odeslani zpravy podle vybraneho (aktivniho) adapteru
switch(m_active)
{
    case 0:
        USB2CAN_SelectActualAdapter(USB2CAN_adapter_1);
        USB2CAN_SendCANMessage(message);
        m_log_usb2can_1.AddString("Message sended");
        PrintMessageInfo(&m_log_usb2can_1, &message);
    break;
    case 1:
        USB2CAN_SelectActualAdapter(USB2CAN_adapter_2);
        USB2CAN_SendCANMessage(message);
        m_log_usb2can_2.AddString("Message sended");
        PrintMessageInfo(&m_log_usb2can_2, &message);
    break;
```

```
    case 2:
        PP2CAN_SendCANMessage(message);
        m_log_pp2can.AddString("Message sended");
        PrintMessageInfo(&m_log_pp2can, &message);
    break;
}
```

Kontrola příchodu zprávy probíhá ve funkci OnTimer takto:

```
// Prijem a zpracovani zprav
CAN_MESSAGE message;
// Vyber aktualniho USB2CAN adapteru
USB2CAN_SelectActualAdapter(USB2CAN_adapter_1);
// Jsou prijaty nejake zpravy prvnim USB2CAN adapterem
if(USB2CAN_GetRXBufferLength()>0)
{
    // Jestlize ano, prectu jednu z nich
    if(USB2CAN_GetCANMessage(&message))
    {
        m_log_usb2can_1.AddString("Message received");
        PrintMessageInfo(&m_log_usb2can_1, &message);
    }
}
// Jsou prijaty nejake zpravy druhym USB2CAN adapterem
USB2CAN_SelectActualAdapter(USB2CAN_adapter_2);
if(USB2CAN_GetRXBufferLength()>0)
{
    if(USB2CAN_GetCANMessage(&message))
    {
        m_log_usb2can_2.AddString("Message received");
        PrintMessageInfo(&m_log_usb2can_2, &message);
    }
}

// Jsou prijaty nejake zpravy PP2CAN adapterem
if(PP2CAN_GetRXBufferLength()>0)
{
    if(PP2CAN_GetCANMessage(&message))
    {
        m_log_pp2can.AddString("Message received");
        PrintMessageInfo(&m_log_pp2can, &message);
    }
}
```

Protože však během časových intervalu, kdy je volána funkce OnTimer může přijít více zpráv, je vhodné přepsat čtení CAN zpráv takto:

```
    while(PP2CAN_GetCANMessage(&message))
    {
        m_log_pp2can.AddString("Message received");
        PrintMessageInfo(&m_log_pp2can, &message);
    }
```

Pokud by vyčítání zpráv probíhalo v samostatném vlákně, je vhodné používat funkcí: PP2CAN_WaitForMessage a USB2CAN_WaitForMessage. V případě práce s více adaptéry USB2CAN ve více vláknech, je třeba uzavírat volání funkcí (skupin funkcí) mezi USB2CAN_SelectActualAdapterAccess a USB2CAN_UnselectActualAdapterAccess.

8. Podrobný popis funkcí API USB2CAN

V současné době je mezi uživateli USB2CAN API nejpoužívanější variantou API X2CAN. Proto následující text obsahuje podrobný popis chování a použití funkcí tohoto API.

```
X2CAN_DLLMAPPING void* USB2CAN_PrepareAdapter(void);
```

Funkce alokuje pamět a inicializuje některé proměnné datové struktury, se kterou pracují funkce

API. Tato funkce je základní a nejjednodušší variantou vytvoření této struktury. Inicializaci této struktury je třeba provést ještě před vlastním otevřením adaptéru. Je určena pro variantu adaptéru bez osazené konfigurační paměti EEPROM. V tomto případě je nastaven defaultní výběr výběr zařízení typu „OPEN_BY_DESCRIPTION“, identifikační řetězec je nastaven na „USB <-> Serial“. Tímto řetězcem se standardně identifikují obvody FTDI 245 bez konfigurační EPROM. Dále je v této funkci inicializována kritická sekce, která slouží k ošetření výlučného přístupu k zařízení v případě vícethreadové aplikace pomocí „USB2CAN_SelectActualAdapterAccess“.

```
X2CAN_DLLMAPPING void* USB2CAN_PrepareAdapterEEPROM(void);
```

Tato funkce je určena pro adaptéry s osazenou EEPROM. Funkce se liší pouze v inicializaci identifikačního řetězce a to na hodnotu „USB2CAN“.

```
X2CAN_DLLMAPPING void* USB2CAN_PrepareAdapterEx(EUSB2CANDevice selector, char
*name)
```

Poslední varianta inicializace datové struktury dovoluje specifikovat výběr adaptéru pomocí parametru Device_selector. Je možno použí tyto varianty: „OPEN_BY_DEVICE_NUMBER, OPEN_BY_SERIAL_NUMBER a OPEN_BY_DESCRIPTION“. Parametr name pak musí obsahovat identifikační řetězec. OPEN_BY_DEVICE_NUMBER dovoluje vybrat adaptér podle čísla připojení, name pak obsahuje například řetězec „0“. OPEN_BY_DESCRIPTION dovoluje použít volby „USB <-> Serial“ nebo „USB2CAN“. Poslední varianta OPEN_BY_SERIAL_NUMBER dovoluje vybrat adaptér dle sériového čísla adaptéru, které je nastaveno v konfigurační EEPROM.

```
X2CAN_DLLMAPPING void USB2CAN_SelectActualAdapter(void* adapter);
```

Tato funkce je používána jen v některých speciálních případech, kdy aplikace nepracuje s adaptérem vícevláknově. Nastavuje adaptér se kterým chceme pracovat, nenastavuje však výhradní přístup k adaptéru pomocí kritické sekce, pouze nastavuje aktuální adaptér se kterým chce uživatel pracovat. Po zavolání této funkce je možné volat funkce pro otevření adaptéru, odeslání zprávy a podobně.

```
X2CAN_DLLMAPPING void USB2CAN_SelectActualAdapterAccess(void* adapter);
```

Nastavuje adaptér se kterým se bude pracovat a uzamyká kritickou sekci pro výhradní přístup. Následně můžeme volat funkci (funkce) pro práci s adaptérem. Po zavolání těchto funkcí musí následovat volání: „USB2CAN_UnselectActualAdapterAccess“. Funkce „USB2CAN_SelectActualAdapterAccess“ je určena primárně pro použitíve vícethreadových aplikacích a při práci s více adaptéry současně, kdy pomocí této funkce měníme adaptér se kterým pracujeme.

```
X2CAN_DLLMAPPING void USB2CAN_UnselectActualAdapterAccess(void);
```

Odemče kritickou sekci, která byla zamčena v „USB2CAN_SelectActualAdapterAccess“.

```
X2CAN_DLLMAPPING void USB2CAN_DestroyAdapter(void* adapter);
```

Po ukončení práce s adaptérem například při ukončení uživatelské aplikace uvolňuje tato funkce pamět alokovanou pro adaptér.

```
X2CAN_DLLMAPPING bool USB2CAN_Open(CAN_SPEED speed, bool CreateCommThreads , void
(*error_function)(int err_code, const char * error_string), bool low_speed);
```

Funkce slouží k otevření a inicializaci adaptéru USB2CAN. Parametr CAN_SPEED nastavuje komunikační rychlost. CreateCommThreads je třeba vždy nastavovat na true. V opačném případě nejsou vytvořena nízkoúrovňová komunikační vlákna nutná pro funkci adaptéru. Ty se nevytváří jen ve speciálních případech například při ladění firmware. Parametr error_function je ukazatel na funkci kterou volá API v případě chyby předává do ní textový popis chyby. Tento ukazatel je možné nastavit na NULL. Poslední parametr low_speed se nastavuje na true v případě použití low_speed adaptéru.

```
X2CAN_DLLMAPPING  USB2CAN_CloseErrorChannel(void);
```

Tato funkce nastavuje ukazatel na chybovou funkci předaný při volání „USB2CAN_Open“ na NULL. Tato funkce je určena například pro případy kdy dochází v dusledku například stavu BusOff k častému volání chybové funkce, což znemožňuje uzavření uživatelské aplikace.

```
X2CAN_DLLMAPPING bool USB2CAN_Close(void);
```

Standardní uzavření CAN portu zařízení USB2CAN.

```
X2CAN_DLLMAPPING bool USB2CAN_CloseExt(void);
```
Tato varianta uzavření CAN portu zařízení USB2CAN provede navíc SW simulaci odpojení a připojení zařízení. Je určena pro případy kdy dojde k závažné chybě USB driveru.

```
X2CAN_DLLMAPPING void USB2CAN_CheckUSBStatus(char *after_action);
```
Funkce je určena primárně pro interní použití USB2CAN API. Kontroluje status USB zařízení, v případě chyby volá error funkci. Text „after_action“ je použit v chybovém hlášení předaném do error funkce k bližší identifikaci místa vzniku chyby.

```
X2CAN_DLLMAPPING bool USB2CAN_CheckUSB(void);
```
Funkce je určena primárně pro interní použití USB2CAN API. Vrací false pokud je signalizován chybný USB status.

```
X2CAN_DLLMAPPING void USB2CAN_SimulateUSBError(unsigned int status);
```
Funkce je určena primárně pro interní použití při vývoji USB2CAN API. Simuluje chybu na USB.

```
X2CAN_DLLMAPPING bool USB2CAN_CyclePort(void);
```
Funkce je určena primárně pro interní použití USB2CAN API. Provede SW simulaci odpojení a připojení USB2CAN zařízení. Vrací true pokud simulace odpojení a připojení proběhla v pořádku.

```
X2CAN_DLLMAPPING bool USB2CAN_WaitForConnection(unsigned int hundreds_of_ms);
```
Funkce čeká na připojení zařízení, které bylo specifikováno při předchozím volání některé z funkcí typu USB2CAN_PrepareAdapter.

```
X2CAN_DLLMAPPING void USB2CAN_SetTimeout(int ms);
```
Funkce je určena primárně pro interní použití USB2CAN API. Uživatelské použití této funkce není doporučeno. Nastavuje timeouty čekání na data z USB2CAN adaptéru.

```
X2CAN_DLLMAPPING void USB2CAN_SetUSBBlockLimit(int Limit);
```
Funkce je určena primárně pro interní použití USB2CAN API. Nastavuje limit při kterém USB2CAN začne blokovat příjem požadavků na odeslání CAN zprávy z důvodu přeplněnosti interních bufferů procesoru. Povoleny hodnoty 1-18. Provádí modifikaci tohoto parametru, neprovádí nastavení v adaptéru USB2CAN.

```
X2CAN_DLLMAPPING void USB2CAN_SetupBasic(int BaudRate);
```
Funkce je určena primárně pro interní použití USB2CAN API. Inicializuje řadič SJA1000, nastavuje komunikační rychlost. Uživatel při stanardním použití API tuto funkci nepoužívá neboť tato funkce se volá při volání USB2CAN_Open.

```
X2CAN_DLLMAPPING void USB2CAN_ListenOnly(int BaudRate, bool listen_only);
```
Funkce dovoluje zapínat/vypínat funkci Listen only. V tomto módu USB2CAN nedovoluje odesílat zprávy, pouze pasívně zachytává zprávy, nijak nezasahuje do komunikace na CAN sběrnici. Více podrobností k tomuto módu lze nalézt v dokuentaci k obvodu SJA1000.

```
X2CAN_DLLMAPPING bool USB2CAN_Loopback();
```
Funkce je určena primárně pro interní použití USB2CAN API. Odesílá zprávu Loopback do zařízení USB2CAN, zařízení toto zprávu zašle zpět do PC. Funkce čeká, dokud neobdrží zprávu zpět nebo nevyprší timeout.

```
X2CAN_DLLMAPPING bool USB2CAN_SetMode(int Mode);
```
Funkce je určena primárně pro interní použití USB2CAN API. Nastaví jeden ze 4 pracovních módů zařízení USB2CAN. Jedná se o módy: BOOT_MODE, CONFIG_MODE, NORMAL_MODE a LOOPBACK_MODE.

```
X2CAN_DLLMAPPING bool USB2CAN_GetMode(int *Mode);
```
Vrací aktuální nastavený mód. Parametr Mode musí ukazovat na platnou proměnnou.

```
X2CAN_DLLMAPPING bool USB2CAN_GetFirmwareVersion(char *Version);
```

Vrací aktuální verzi firmware zařízení USB2CAN. Paramter Version musí ukazovat na pole o minimální délce 13 znaků.

```
X2CAN_DLLMAPPING bool USB2CAN_Command0(unsigned char Command);
```

Funkce je určena primárně pro interní použití USB2CAN API. Dovoluje odesílat příkazy typu Command do zařízení USB2CAN. Více podrobností o těchto příkazech lze nalézt v dokumentu: USB2CAN: Struktura USB komunikace.

```
X2CAN_DLLMAPPING bool USB2CAN_Command1(unsigned char Command, unsigned char
Param1);
```

Funkce je určena primárně pro interní použití USB2CAN API. Dovoluje odesílat příkazy typu Command do zařízení USB2CAN. Více podrobností o těchto příkazech lze nalézt v dokumentu: USB2CAN: Struktura USB komunikace.

```
X2CAN_DLLMAPPING bool USB2CAN_Command2(unsigned char Command, unsigned char Param1,
unsigned char Param2);
```

Funkce je určena primárně pro interní použití USB2CAN API. Dovoluje odesílat příkazy typu Command do zařízení USB2CAN. Více podrobností o těchto příkazech lze nalézt v dokumentu: USB2CAN: Struktura USB komunikace.

```
X2CAN_DLLMAPPING bool USB2CAN_Command3(unsigned char Command, unsigned char Param1,
unsigned char Param2, unsigned char Param3);
```

Funkce je určena primárně pro interní použití USB2CAN API. Dovoluje odesílat příkazy typu Command do zařízení USB2CAN. Více podrobností o těchto příkazech lze nalézt v dokumentu: USB2CAN: Struktura USB komunikace.

```
X2CAN_DLLMAPPING bool USB2CAN_ReadReg(unsigned char Address, unsigned char *Data);
```

Funkce je určena primárně pro interní použití USB2CAN API. Je určena ke čtení zadaného registru CAN řadiče SJA1000.

```
X2CAN_DLLMAPPING bool USB2CAN_WriteReg(unsigned char Address, unsigned char Data);
```

Funkce je určena primárně pro interní použití USB2CAN API. Je určena k zápisu do zadaného registru CAN řadiče SJA1000.

```
X2CAN_DLLMAPPING bool USB2CAN_WriteReadReg(unsigned char Address, unsigned char
Data, unsigned char *DataOut);
```

Funkce je určena primárně pro interní použití USB2CAN API. Je určena k zápisu do zadaného registru CAN řadiče SJA1000. Po provedení zápisu provede čtení tohoto registru.

```
X2CAN_DLLMAPPING bool USB2CAN_BitModReg(unsigned char Address, unsigned char Mask,
unsigned char Data);
```

Funkce je určena primárně pro interní použití USB2CAN API. Provádí modifikaci bitů registru dle zadané masky. Jsou modifikovány bity které odpovídají bitů s hodnotou 1 v masce.

```
X2CAN_DLLMAPPING bool USB2CAN_BitModReadReg(unsigned char Address, unsigned char
Mask, unsigned char Data, unsigned char *DataOut);
```

Funkce je určena primárně pro interní použití USB2CAN API. Provádí modifikaci bitů registru dle zadané masky. Jsou modifikovány bity které odpovídají bitů s hodnotou 1 v masce. Po ukončení modifikace provede zpětné čtení hodnoty registru.

```
X2CAN_DLLMAPPING bool USB2CAN_WriteInstruction(unsigned __int16 Address, unsigned
__int16 Instruction);
```

Funkce je určena primárně pro interní použití USB2CAN API. Slouží k modifikaci firmware adaptéru USB2CAN. Avšak v uživatelských verzích API je funkce nevykoná žádnou činnost.

```
X2CAN_DLLMAPPING bool USB2CAN_ReadInstruction(unsigned __int16 Address, unsigned
__int16 *Instruction);
```

Funkce je určena primárně pro interní použití USB2CAN API. Slouží k modifikaci firmware adaptéru USB2CAN. Avšak v uživatelských verzích API je funkce nevykoná žádnou činnost.

```
X2CAN_DLLMAPPING bool USB2CAN_ReadTEC(unsigned char *TEC);
```
Funkce čte hodnotu registru TEC (Transmit Error Counter). Interně pracuje tak že zašle požadavek na čtení tohoto registru a čeká na jeho dokončení. Tím blokuje ostatní transakce na USB stímto adaptérem. Proto je doporučeno čtení tohoto registru pomocí kombinace volání „USB2CAN_EnableReadTEC“ a „USB2CAN_GetTEC“ kdy k blokování transakcí nedochází, o čtení se stará vnitřní vrstva API.

```
X2CAN_DLLMAPPING bool USB2CAN_ReadREC(unsigned char *REC);
```
Funkce čte hodnotu registru REC (Receive Error Counter). Interně pracuje tak že zašle požadavek na čtení tohoto registru a čeká na jeho dokončení. Tím blokuje ostatní transakce na USB stímto adaptérem. Proto je doporučeno čtení tohoto registru pomocí kombinace volání „USB2CAN_EnableReadREC“ a „USB2CAN_GetREC“ kdy k blokování transakcí nedochází, o čtení se stará vnitřní vrstva API.

```
X2CAN_DLLMAPPING bool USB2CAN_ReadRST(unsigned int *RST);
```
Funkce čte hodnotu Reset counteru. Tento čítač počítá resety obvodu SJA1000 při přechodu do stavu BusOff.

```
X2CAN_DLLMAPPING void USB2CAN_SetPassiveMode(bool mode);
```
Nastavuje pasivní komunikační mód. Blokuje odeslání zprávy na CAN ve funkcích API aby bylo zabráněno náhodnému odelání CAN zprávy při práci na neznámé CAN systému.

```
X2CAN_DLLMAPPING void USB2CAN_SetOneShotMode(bool mode);
```
Nastavuje mód „One Shot“. V tomto módu pokud se nepodaří řadiči CANu zprávu odeslat, nebude provádět další pokus o odeslání.

```
X2CAN_DLLMAPPING void USB2CAN_CmdBasicCAN(void);
```
Funkce je určena primárně pro interní použití USB2CAN API. Natavuje mód BasicCAN řadiče SJA1000. Uživatelské použití této funkce není doporučeno.

```
X2CAN_DLLMAPPING void USB2CAN_CmdResetMode(void);
```
Funkce je určena primárně pro interní použití USB2CAN API. Natavuje Reset mód řadiče SJA1000. Uživatelské použití této funkce není doporučeno.

```
X2CAN_DLLMAPPING void USB2CAN_CmdPeliCAN(void);
```
Funkce je určena primárně pro interní použití USB2CAN API. Natavuje PeliCAN mód řadiče SJA1000. Uživatelské použití této funkce není doporučeno.

```
X2CAN_DLLMAPPING void USB2CAN_CmdOperatingMode(void);
```
Funkce je určena primárně pro interní použití USB2CAN API. Natavuje Operační mód řadiče SJA1000. Uživatelské použití této funkce není doporučeno.

```
X2CAN_DLLMAPPING void USB2CAN_CmdBaudRate(unsigned char t0, unsigned char t1);
```
Funkce je určena primárně pro interní použití USB2CAN API. Nastavuje Timing registry řadiče SJA1000.

```
X2CAN_DLLMAPPING void USB2CAN_CmdCriticalTransmitLimit(unsigned char Limit);
```
Funkce je určena primárně pro interní použití USB2CAN API. Nastavuje limit při kterém USB2CAN začne blokovat příjem požadavků na odeslání CAN zprávy z důvodu přeplněnosti interních bufferů procesoru. Povoleny hodnoty 1-18. Provádí nastavení parametru v adaptéru USB2CAN.

```
X2CAN_DLLMAPPING void USB2CAN_CmdReadyTransmitLimit(unsigned char Limit);
```
Funkce je určena primárně pro interní použití USB2CAN API. Nastavuje limit při kterém USB2CAN začne vypne blokování příjmu požadavků na odeslání CAN zprávy z důvodu přeplněnosti interních bufferů procesoru. Povoleny hodnoty 1-18. Provádí nastavení parametru v adaptéru USB2CAN.

```
void USB2CAN_HandBasicCAN(void);
```

Funkce je určena primárně pro interní použití USB2CAN API. Na rozdíl od podobné již dříve uvedené funkce, která zasílá požadavek na nastavení módu do zařízení USB2CAN nastavuje tato funkce mód BasicCAN nastavením jednotlivých registrů z PC.

```
void USB2CAN_HandPeliCAN(void);
```
Funkce je určena primárně pro interní použití USB2CAN API. Na rozdíl od podobné již dříve uvedené funkce, která zasílá požadavek na nastavení módu do zařízení USB2CAN nastavuje tato funkce mód PeliCAN nastavením jednotlivých registrů z PC.

```
void USB2CAN_HandResetMode(void);
```
Funkce je určena primárně pro interní použití USB2CAN API. Na rozdíl od podobné již dříve uvedené funkce, která zasílá požadavek na nastavení módu do zařízení USB2CAN nastavuje tato funkce Reset mód nastavením jednotlivých registrů z PC.

```
void USB2CAN_HandOperatingMode(void);
```
Funkce je určena primárně pro interní použití USB2CAN API. Na rozdíl od podobné již dříve uvedené funkce, která zasílá požadavek na nastavení módu do zařízení USB2CAN nastavuje tato funkce Operating mód nastavením jednotlivých registrů z PC.

```
void USB2CAN_HandListenOnlyMode(void);
```
Funkce je určena primárně pro interní použití USB2CAN API. Na rozdíl od podobné již dříve uvedené funkce, která zasílá požadavek na nastavení módu do zařízení USB2CAN nastavuje tato funkce ListenOnly mód nastavením jednotlivých registrů z PC.

```
X2CAN_DLLMAPPING void USB2CAN_BaudRate_Osc16(int BaudRate);
```
Funkce je určena primárně pro interní použití USB2CAN API. Nastavuje hodnoty Timing registrů v datové struktůře adaptéru. Neprovádí však nastavení v zařízení USB2CAN.

```
X2CAN_DLLMAPPING void USB2CAN_BaudRate_Osc20(int BaudRate);
```
Funkce je určena primárně pro interní použití USB2CAN API. Nastavuje hodnoty Timing registrů v datové struktůře adaptéru. Neprovádí však nastavení v zařízení USB2CAN. Funkce je určena pouze pro speciální variantu adaptéru osazenou krystalem 20MHz.

```
bool USB2CAN_SendUSBLoopback();
```
Funkce je určena primárně pro interní použití USB2CAN API. Odesílá zprávu Loopback do zařízení USB2CAN. Na rozdíl od funkce USB2CAN_Loopback nečeká na přijetí této zprávy zpět. Funkce je dostupná jen při použití statické knihovny.

```
bool USB2CAN_SendUSB(unsigned char *data, int length);
```
Funkce určena pouze pro vývoj a testování. Funkce je dostupná jen při použití statické knihovny.

```
bool USB2CAN_SendUSBReadReg(unsigned char Address);
```
Funkce určena pouze pro vývoj a testování. Funkce je dostupná jen při použití statické knihovny.

```
bool USB2CAN_SendUSBWriteReg(unsigned char Address, unsigned char data);
```
Funkce určena pouze pro vývoj a testování. Funkce je dostupná jen při použití statické knihovny.

```
X2CAN_DLLMAPPING void USB2CAN_GetStatistics(USB2CAN_Statistics *Statistics);
```
Funkce určena pouze pro vývoj a testování. Na adresu struktury Statistics vrací statistiku o průběju komunikace s adaptérem USB2CAN a sběrnicí CAN.

```
double USB2CAN_PerformanceTest(int NumberOfMessages);
```
Funkce určena pouze pro vývoj a testování. Funkce je dostupná jen při použití statické knihovny.

```
double USB2CAN_PerformanceReadTest(int NumberOfMessages);
```
Funkce určena pouze pro vývoj a testování. Funkce je dostupná jen při použití statické knihovny.

```
double USB2CAN_PerformanceWriteTest(int NumberOfMessages);
```
Funkce určena pouze pro vývoj a testování. Funkce je dostupná jen při použití statické knihovny.

```
X2CAN_DLLMAPPING void USB2CAN_SWSetTransmitCritical(bool state);
```
Funkce určena pouze pro vývoj a testování. Simuluje zaplnění interních bufferů zařízení USB2CAN.

```
X2CAN_DLLMAPPING int USB2CAN_GetTransmitErrors();
```
Funkce určena pouze pro vývoj a testování.

```
X2CAN_DLLMAPPING void USB2CAN_ClearTransmitErrors();
```
Funkce určena pouze pro vývoj a testování.

```
X2CAN_DLLMAPPING int USB2CAN_GetTXBufferLength();
```
Vrací velikost softwarového bufferu zpráv v PC, které čekají na odeslání na CAN.

```
X2CAN_DLLMAPPING int USB2CAN_GetRXBufferLength();
```
Vrací velikost softwarového bufferu zpráv v PC přijatých z CANu, které čekají na zpracování.

```
X2CAN_DLLMAPPING void USB2CAN_SetTimeStampMode (bool mode);
```
Povoluje měření času přijetí zprávy. Čas je nastaven u každé zprávy zvlášť při jejím přijetí. Vzrůstá ovšem zátěž procesoru PC.

```
X2CAN_DLLMAPPING void USB2CAN_ClearBuffers (void);
```
Vymaže obsah bufferů zpráv čekajících na odesání na CAN i přijatých z CANu čekajících na zpracování.

```
X2CAN_DLLMAPPING unsigned char USB2CAN_GetREC (void);
```
Funkce je určena pro čtení registru REC (Receive Error Counter) z CAN řadiče SJA1000. Čtení musí být povoleno funkcí „USB2CAN_EnableReadREC“.

```
X2CAN_DLLMAPPING unsigned char USB2CAN_GetTEC (void);
```
Funkce je určena pro čtení registru TEC (Transmit Error Counter) z CAN řadiče SJA1000. Čtení musí být povoleno funkcí „USB2CAN_EnableReadREC“.

```
X2CAN_DLLMAPPING unsigned int USB2CAN_GetRST (void);
```
Funkce je určena pro čtení čítače RST (Reset Counter).

```
X2CAN_DLLMAPPING void USB2CAN_EnableReadTEC (bool enable);
```
Povoluje čtení registru TEC. O periodické čtení se stará API. Aktuální hodnot lze číst pomocí „USB2CAN_GetTEC“.

```
X2CAN_DLLMAPPING void USB2CAN_EnableReadREC (bool enable);
```
Povoluje čtení registru REC. O periodické čtení se stará API. Aktuální hodnot lze číst pomocí „USB2CAN_GetREC“.

```
X2CAN_DLLMAPPING void USB2CAN_AutoRecconectEnable(bool enable);
```
Funkce je určena primárně pro interní použití USB2CAN API.

```
X2CAN_DLLMAPPING void USB2CAN_HighBusLoad(void);
```
Zvyšuje prioritu vlákna pro čtení dat z USB.

```
X2CAN_DLLMAPPING void USB2CAN_LowBusLoad(void);
```
Snižuje prioritu vlákna pro čtení dat z USB. Pro případy kdy chceme snížit zátěž PC a ne CANu není velký „provoz“.

```
X2CAN_DLLMAPPING void USB2CAN_LowTransmit(void);
```
Snižuje prioritu vlákna pro odesílání CAN zpráv v případě že preferujeme příjem zpráv a odesílání zpráv je prováděno minimálně nebo vůbec.

```
X2CAN_DLLMAPPING void USB2CAN_ImproveEMCImmunity(unsigned int count);
```

Zvyšuje odolnost USB na rušení. Interně volá funkci FTDI driveru FT_SetResetPipeRetryCount.

```
X2CAN_DLLMAPPING bool USB2CAN_SendCANMessage(CAN_MESSAGE message);
```

Funkce je určena pro odeslání CAN zprávy. Zpráva je předána strukturou CAN_MESSAGE. API je orientováno především na práci se zprávami v tomto formátu. Předpokládá se nastavení položek Id1 (standardní, 11 bitová část identifikátoru) a Id2 (rozšířená, 18 bitová část identifikátoru). Pokud pracujete s 29 bitovým identifikátorem Id, je třeba aktualizovat Id1 a Id2 pomocí funkce „CANMsgUpdateFrom29“ (canbus.h).

```
X2CAN_DLLMAPPING bool USB2CAN_SendRegisterMessage(unsigned char Data[13]);
```

Funkce je určena pro odeslání CAN zprávy. Zpráva je zadána ve formátu registrů obvodu SJA1000.

```
X2CAN_DLLMAPPING bool USB2CAN_SendSJAMessage(SJA1000MsgData *Data);
```

Funkce je určena pro odeslání CAN zprávy. Zpráva je zadána ve formátu registrů obvodu SJA1000. Není však použito pole bajtů ale struktura „SJA1000MsgData“ která dovoluje lepší orientaci v jednotlivých bajtech.

```
X2CAN_DLLMAPPING bool USB2CAN_SendStandardMessage(unsigned __int16 StandardId, bool
RTR, unsigned char Length, unsigned char *Data);
```

Funkce je určena pro odeslání standardní, 11 bitové CAN zprávy. Jednotlivé položky zprávy jsou zadávány smostatně, není tak třeba vyplňovat strukturu CAN_MESSAGE.

```
X2CAN_DLLMAPPING bool USB2CAN_SendExtendedMessage(unsigned __int16 StandardId,
unsigned __int32 ExtendedId, bool RTR, unsigned char Length, unsigned char *Data);
```

Funkce je určena pro odeslání rozšířené, 29 bitové CAN zprávy. Jednotlivé položky zprávy jsou zadávány smostatně, není tak třeba vyplňovat strukturu CAN_MESSAGE. Standardní i rozšířená část jsou zadány samostatně.

```
X2CAN_DLLMAPPING bool USB2CAN_SendExtendedMessage29(unsigned __int32 ExtendedId,
bool RTR, unsigned char Length, unsigned char *Data);
```

Funkce je určena pro odeslání rozšířené, 29 bitové CAN zprávy. Jednotlivé položky zprávy jsou zadávány smostatně, není tak třeba vyplňovat strukturu CAN_MESSAGE. Identifikátor je zadán v 29 bitovém formátu.

```
X2CAN_DLLMAPPING bool USB2CAN_SimulateCANMessage(CAN_MESSAGE message);
```

Funkce dovoluje z PC simulovat příchod zprávy.

```
X2CAN_DLLMAPPING bool USB2CAN_GetSJAMessage(SJA1000MsgData *data);
```

Vrací true, je-li zpráva přečtena, pokud je bufer přijatých zpráv prázdný, vrací false. Zpráva je uložena na adresu předanou parametrem „data“ ve formátu registrů obvodu SJA1000. Adresa struktury „SJA1000MsgData“ musí tedy platná.

```
X2CAN_DLLMAPPING bool USB2CAN_GetCANMessage(CAN_MESSAGE *message);
```

Vrací true, je-li zpráva přečtena, pokud je bufer přijatých zpráv prázdný, vrací false. Zpráva je uložena na adresu předané parametrem „message“ ve formátu registrů CAN_MESSAGE. Adresa struktury „CAN_MESSAGE“ musí tedy platná.

```
X2CAN_DLLMAPPING bool USB2CAN_GetMessage11_18(bool *StExt, unsigned __int16
*StandardId, unsigned __int32 *ExtendedId, bool *RTR, unsigned char *Length,
unsigned char *Data);
```

Vrací true, je-li zpráva přečtena, pokud je bufer přijatých zpráv prázdný, vrací false. Zpráva je uložena na adresy předané parametry, všechny ukazatele musejí tedy být platné. Zpráva je čtena ve formátu 11+(18) bitů.

```
X2CAN_DLLMAPPING bool USB2CAN_GetMessage11_18_t(bool* StExt, unsigned __int16*
StandardId, unsigned __int32* ExtendedId, bool* RTR, unsigned char* Length,
unsigned char* Data, CANMessageTime* time);
```

Vrací true, je-li zpráva přečtena, pokud je bufer přijatých zpráv prázdný, vrací false. Zpráva je uložena na adresy předané parametry, všechny ukazatele musejí tedy být platné. Zpráva je čtena ve

formátu 11+(18) bitů. Na rozdíl od předešlé funkce nastavuje na adresu předanou parametrem CANMessageTime čas příjmu zprávy.

```
X2CAN_DLLMAPPING bool USB2CAN_GetMessage29(bool *StExt, unsigned __int32 *Id, bool
*RTR, unsigned char *Length, unsigned char *Data);
```

Vrací true, je-li zpráva přečtena, pokud je bufer přijatých zpráv prázdný, vrací false. Zpráva je uložena na adresy předané parametry, všechny ukazatele musejí tedy být platné. Zpráva je čtena ve formátu 29 bitů.

```
X2CAN_DLLMAPPING bool USB2CAN_WaitForRxMessage(unsigned int timeout);
```

Funkce čeká na příchod zprávy, maimálně však po zadaný počet milisekund. Byla li zpráva přijata, vrací true.

```
X2CAN_DLLMAPPING bool USB2CAN_WaitForRxMessage2(unsigned int timeout,void
*adapter);
```

Funkce čeká na příchod zprávy, maimálně však po zadaný počet milisekund. Byla li zpráva přijata, vrací true. Tato varianta je vhodná pro případy, kdy pracujete s více adaptéry. Do funkce se předává ukazatel na datovou strukturu adaptéru, nevyžaduje nastavení aktuálního adaptéru pomocí „USB2CAN_SelectActualAdapterAccess“.

```
X2CAN_DLLMAPPING char* USB2CAN_GetUSBDeviceInfo(void);
```

Funkce vrací ukazatel na textový řetězec se seriovým číslem adaptéru.

```
X2CAN_DLLMAPPING bool USB2CAN_ErrorCaptureDecoder(unsigned char ECC_register, char*
TextDescription);
```

Funkce je určena pro dekódování chybového registru ECC. Textový popis chyby je uložen na adresu předanou parametrem TextDescription.

9. Změny ve verzích API

2.025	- Přidáno rozhraní pro práci s adaptérem prostřednictvím sítě a Remote CAN serveru. - Přidána možnost zaregistrovat funkce pro práci s adaptéry jiných výrobců.
2.020	- Funkce jsou deklarovány s Extern “C“. Tato úprava byla provedena pro lepší přenositelnost DLL mezi překladači. To si vyžádalo přejmenování některých funkcí, které byly původně přetíženy (například funkce USB2CAN_GetMessage11_18). - Funkce USB2CAN_PrepareAdapter je určena pro adaptér bez konfigrační EEPROM, pro adaptér s konfigurační EEPROM je určena funkce USB2CAN_PrepareAdapterEEPROM. - Funkce USB2CAN_GetUSBDeviceInfo vrací sériové číslo adaptéru. - Funkce USB2CAN_HighBusLoad, USB2CAN_LowBusLoad a USB2CAN_LowTransmit dovolují v některých aplikacích snížit zátěž CPU.
2.015	- Funkce pro zabezpečení výhradního přístupu k adaptérům při současné práci s více adaptéry: USB2CAN_SelectActualAdapterAccess a USB2CAN_SelectActualAdapterAccess. Funkce USB2CAN_ErrorCaptureDecoder pro dekódování chyb přenosu na CANu. - CANMsgUpdateFrom11_18 a CANMsgUpdateFrom29 pro synchronizaci tvaru identifikátorů - Přidány funkce pro vytváření identifikátorů některých high-level protokolů (např. CANMsgSAE).