Informační a řídicí systémy I.
Ovladače v OS a v ŘS REX

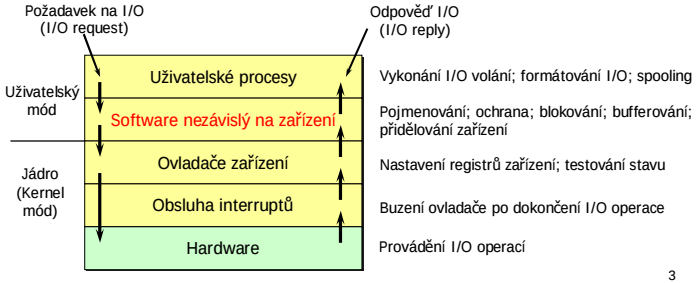
Pavel Balda
ZČU v Plzni, FAV, KKY

Osnova přednášky

- Komunikace s ovladači ve Windows (Win32)
  - Funkce pro práci se soubory
  - DeviceIoControl()
- Ovladače ŘS REX
  - Typy ovladačů ŘS REX
  - Funkce pro HOST část ovladače
  - Funkce pro TARGET část ovladače

Komunikace s ovladači v OS

- Typické schéma jednotlivých vrstev v I/O subsystému OS je na obrázku
- Spolupráce mezi uživatelskými programy a ovladači zařízení je realizována přes vrstvu Softwaru nezávislého na zařízení
- Tato vrstva sjednocuje přístup k různým zařízení pomocí abstrakce



Ovladače ve Windows

- Většina ovladačů pro Windows je zařazena do jádra (Windows kernel)
- Vývoj ovladačů do Windows je náročná práce, vyžadující velkou zkušenost s OS Windows
- Windows Driver Foundation (WDF) – vývojová skupina, vytvářející nástroje pro budování ovladačů pro Windows 2003, XP, Server 2003, Vista, a novější
- WDF pracuje na Windows Driver Framework – nový model pro vývoj ovladačů do Windows. Má dvě varianty:
  - Kernel-Mode Driver Framework (KMDF) – pro tvorbu standardních ovladačů jádra (většina zařízení) – založeno na API v jazyku C, je součástí WDK
  - User-Mode Driver Framework (UMDF) – pro tvorbu tříd ovladačů speciálních zařízení založených na komunikačních protokolech (např. kamery, přehrávače, apod.) v uživatelském módu – založeno na rozhraní COM
- Pro vývoj ovladačů je určen Windows Driver Kit (WDK). Skládá se z:
  - Windows Driver Development Kit (DDK) – tradiční prostředí pro vývoj ovladačů
  - Driver Test Manager (DTM) – soubor testů pro „Windows Logo Program“
- Vývoj ovladačů přesahuje možnosti této přednášky. Více informací lze nalézt na Windows Hardware Developer Central: www.microsoft.com/whdc

Komunikace s ovladači ve Windows

- Vrstva Softwaru nezávislého na zařízení (uživatelský mód) je mapována do funkcí pro práci se souborovým systémem v rozhraní Win32
- Umožňuje komunikovat jak s existujícími ovladači pro Windows (od třetích stran), tak i s vlastními ovladači
- Funkce z Win32 se dají snadno volat z jazyka C/C++
- Dosud není přímá podpora z tříd .NET Framework (do verze 2), tj. ani z C# !
- Pro volání z C# lze použít techniku P-Invoke (Platform Invoke) – import funkcí ze systémových DLL pomocí atributu **`DllImport`**. Viz příklady dále.
- Se ovladači se spolupracuje prostřednictvím tzv. handle (někdy překládán jako madlo či rukojeť **J**)

Nejčastěji používané funkce

- Nejčastěji používané funkce z Win32 pro práci s ovladači zařízení
  - **`CreateFile()`** – vytváření a otvírání souborů, otvírání zařízení
  - **`CloseHandle()`** – zavírání handlů souborů a zařízení
  - **`ReadFile()`** – sekvenční čtení dat ze souborů a z komunikačních zařízení (např. sériových linek)
  - **`WriteFile()`** – zápis dat do souborů a do komunikačních zařízení
  - **`GetLastError()`** – funkce pro vrácení kódu poslední chyby (pro daný thread)
  - **`DeviceIoControl()`** – obecná funkce pro vykonání konkrétní operace ovladačem
- Podrobnou dokumentaci ke všem funkcím lze nalézt na **`http://msdn2.microsoft.com/en-us/library`**

Funkce CreateFile() (1/4)

- Vytváří nebo otvírá daný soubor nebo otvírá práci s daným zařízením

```
HANDLE CreateFile(
   LPCTSTR lpFileName,     // ukazatel na jméno souboru
   DWORD dwDesiredAccess, // přístupový mód (read-write)
   DWORD dwShareMode,      // share mode
   LPSECURITY_ATTRIBUTES lpSecurityAttributes,
                           // ukazatel na atributy zabezpečení
   DWORD dwCreationDisposition, // jak vytvořit?
   DWORD dwFlagsAndAttributes,  // atributy souboru
   HANDLE hTemplateFile    // handle souboru, jehož atributy
);                         // mají být zkopírovány
```

- **`lpFileName`** – název souboru nebo zařízení a k němu příslušného ovladače. Pro zařízení se parametr zadává ve tvaru: **`\\.\DeviceName`**. Příklady:
  - Disketová jednotka A: **`"\\\\.\\a:"`**
  - Fyzický disk 0: **`"\\\\.\\PhysicalDrive0"`**
  - Sériový port: **`"COM1:"`** nebo s vyšším číslem než 9: **`"\\\\.\\COM10"`**
  - Pozor! Při zápisu v řetězci v C/C++ nebo C# (bez uvození znakem @) je třeba zdvojit znaky \

Funkce CreateFile() (2/4)

- **`dwDesiredAccess`** – specifikuje způsob přístupu k zařízení
  - **`0`** – zjišťování atributů k zařízení bez přístupu k němu
  - **`GENERIC_READ`** – data mohou být čtena
  - **`GENERIC_WRITE`** – data mohou být zapisována
- **`dwShareMode`** – bitové příznaky určující, jak může být objekt sdílen. Mohou nabývat bitové kombinace hodnot: **`0`**, **`FILE_SHARE_DELETE`**, **`FILE_SHARE_READ`** nebo **`FILE_SHARE_WRITE`**
- **`lpSecurityAttributes`** – ukazatel na datovou strukturu **`SECURITY_ATTRIBUTES`**, která určuje, zda daný handle může být děděn dceřinými procesy. Je-li **`NULL`**, pak děděn být nemůže.
- **`dwCreationDisposition`** – určuje, jaké akce se mají provést pokud soubor existuje nebo neexistuje. Nabývá jednu z hodnot: **`CREATE_NEW`**, **`CREATE_ALWAYS`**, **`OPEN_EXISTING`**, **`OPEN_ALWAYS`**, **`TRUNCATE_EXISTING`**

Funkce CreateFile() (3/4)

- **dwFlagsAndAttributes** – specifikuje příznaky a atributy souboru
  - Atributy mohou nabývat kombinace hodnot **FILE_ATTRIBUTE_ARCHIVE**, **FILE_ATTRIBUTE_HIDDEN**, **FILE_ATTRIBUTE_NORMAL**, **FILE_ATTRIBUTE_OFFLINE**, **FILE_ATTRIBUTE_READONLY**, **FILE_ATTRIBUTE_SYSTEM**, **FILE_ATTRIBUTE_TEMPORARY**.
  - Všechny atributy lze bitově kombinovat, kromě atributu **FILE_ATTRIBUTE_NORMAL**, který musí být užíván samostatně.
  - Kromě toho lze atributy kombinovat s řadou příznaků **FILE_FLAG_<XXXX>**, podrobně v uživatelské dokumentaci.
- **hTemplateFile** – handle souboru s přístupem **GENERIC_READ**, který bude použit jako vzor pro atributy právě vytvářeného souboru. Parametr může být **NULL**

Funkce CreateFile() (4/4)

- Při otevírání handlu k ovladači by měly být parametry **CreateFile()** nastaveny následovně:
  - **dwDesiredAccess** by měl být nastaven na **FILE_SHARE_READ | FILE_SHARE_WRITE**
  - Pro komunikační zařízení (např. sériové porty) musí být zvolen exklusivní přístup, tj. **dwShareMode** je nastaven na **0**
  - **fdwCreationDisposition** musí mít příznak **OPEN_EXISTING**
  - **hTemplateFile** musí být **NULL**
  - **dwFlagsAndAttributes** může obsahovat **FILE_FLAG_OVERLAPPED**, což značí, že vrácený handle může být použit pro asynchronní (overlapped) operace.
- Poznámka: Pro jednoduchou práci se soubory lze místo **CreateFile()** používat **fopen()**, která však není podporována ve Windows CE!

Funkce CloseHandle()

- Zavírá otevřený handle objektu

```
BOOL CloseHandle(
   HANDLE hObject // handle zavíraného objektu
);
```

- **hObject** – handle k otevřenému objektu, kterým může být:
  - Soubor
  - Ovladač zařízení, komunikační zařízení
  - Proces nebo thread
  - Synchronizační objekt (mutex, semafor, event)
  - A další ...

Funkce ReadFile()

- Čte data ze souboru, komunikačního zařízení nebo socketu

```
BOOL ReadFile(
  HANDLE hFile,     // handle čteného souboru
  LPVOID lpBuffer, // ukazatel na pole přijímaných dat
  DWORD nNumberOfBytesToRead, // požadovaný počet bajtů
  LPDWORD lpNumberOfBytesRead,// skutečný počet přečtených bajtů
  LPOVERLAPPED lpOverlapped   // ukazatel na strukturu OVERLAPPED
);
```

Funkce WriteFile()

- Zapisuje data do souboru, komunikačního zařízení nebo socketu

```
BOOL WriteFile(
  HANDLE hFile,      // handle zapisovaného souboru
  LPCVOID lpBuffer, // ukazatel na pole zapisovaných dat
  DWORD nNumberOfBytesToWrite,    // požadovaný počet bajtů
  LPDWORD lpNumberOfBytesWritten, // počet skutečně zapsaných
                                  //   bajtů
  LPOVERLAPPED lpOverlapped   // ukazatel na strukturu OVERLAPPED
);
```

- Poznámka: Funce **ReadFile()** i **WriteFile()** jsou navrženy jak pro synchronní, tak i asynchronní (overlapped) čtení a zápis. V případě synchronního čtení je parametr **lpOverlapped** roven **NULL**

Funkce GetLastError()

- Vrací kód poslední chyby systémové funkce z volané z daného threadu

```
DWORD GetLastError(VOID)
```

- Je rozumné ji volat vždy po selhání některé jiné funkce pro upřesnění chyby – příčiny selhání.

Funkce DeviceIoControl() (1/2)

- Funkce posílá do ovladače řídicí kód, který způsobí, že zařízení vykoná operaci odpovídající tomuto kódu

```
BOOL DeviceIoControl(
  HANDLE hDevice,           // handle požadovaného zařízení
  DWORD dwIoControlCode,    // řídicí kód požadované operace
  LPVOID lpInBuffer,        // ukazatel na vstupní data operace
  DWORD nInBufferSize,      // velikost vstupních dat v bajtech
  LPVOID lpOutBuffer,       // ukazatel na buffer, do kterého
                            // budou uložena výstupní data
  DWORD nOutBufferSize,     // velikost výstupních bufferu
  LPDWORD lpBytesReturned,  // ukazatel na proměnnou, do níž
                            // bude uložen počet přijatých bajtů
  LPOVERLAPPED lpOverlapped // ukazatel na strukturu pro
);                          // asynchronní operaci
```


Funkce DeviceIoControl() (2/2)

- **dwIoControlCode** – kód, určující jaká operace bude v ovladači provedena
  - Řídicí kód je parametrem určujícím význam následných parametrů funkce DeviceIoControl().
  - Pro různé řídicí kódy mají parametry ***lpInBuffer***, ***nInBufferSize***, ***lpOutBuffer***, a ***nOutBufferSize*** různé významy, pro něž jsou obvykle definovány různé datové struktury, jejichž adresy se po přetypování předávají jako parametry ***lpInBuffer*** a ***lpOutBuffer*** a v parametrech ***nInBufferSize*** a ***nOutBufferSize*** se předávají velikosti těchto struktur určované pomocí operátoru sizeof().
  - Tímto způsobem získává funkce **DeviceIoControl()** výjimečné postavení, neboť může pracovat jako celá množina funkcí.
  - Celá řada kódů je v systémech Windows již předdefinována jako konstanty, jejich názvy mají tvar **IOCTL_<XXXX>** (pro obecné vstupně výstupní operace) nebo **FSCTL_<YYYY>** (pro speciální operace souborového systému)

Standardní kódy IOCTL

- V systému Windows existuje několik desítek předdefinovaných kódů **IOCTL_<XXXX>**
- Řídicí kódy jsou rozděleny do skupin:
  - Komunikační kódy
  - Kódy pro správu zařízení (device management)
    - Např. **IOCTL_STORAGE_EJECT_MEDIA**, **IOCTL_STORAGE_LOAD_MEDIA**, apod.
  - Kódy pro správu adresářů (directory management)
  - Kódy pro správu disků (disk management)
    - Např. **IOCTL_DISK_GET_DRIVE_GEOMETRY_EX**, **IOCTL_DISK_GET_PARTITION_INFO_EX**, apod.
  - Kódy pro správu souborů (file management)
  - Kódy pro správu napájení (power management)
  - Kódy pro správu svazků (volume management)
- Uvedené kódy budou ukázány na příkladech

Import funkce DeviceIoContro() do C#

```
// C#
[DllImport("Kernel32.dll",
CharSet = CharSet.Auto,
SetLastError = true)]
public static extern
bool DeviceIoControl(
  int hDevice,
  int dwIoControlCode,
  byte[] InBuffer,
  int nInBufferSize,
  byte[] OutBuffer,
  int nOutBufferSize,
  ref int pBytesReturned,
  int pOverlapped
);
```

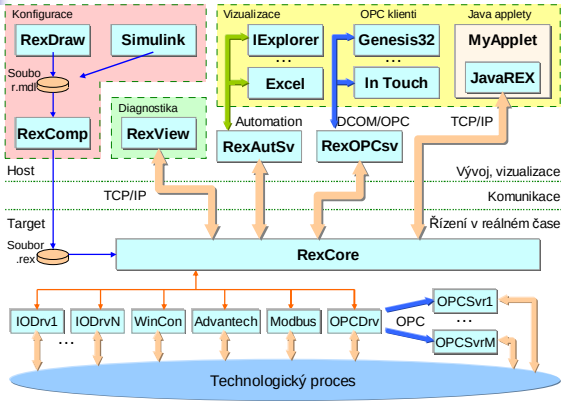
```
// C++
BOOL DeviceIoControl(
  HANDLE hDevice,
  DWORD dwIoControlCode,
  LPVOID lpInBuffer,
  DWORD nInBufferSize,
  LPVOID lpOutBuffer,
  DWORD nOutBufferSize,
  LPDWORD lpBytesReturned,
  LPOVERLAPPED lpOverlapped
);
```

- Více práce dá převod vstupních (výstupních) parametrů do vstupního bufferu **InBuffer** (z výstupního bufferu **OutBuffer**) do potřebných datových struktur

Příklady volání funkce DeviceIoControl()

- Příklady byly původně vytvořeny v prostředí C++ a pak převedeny do C#. Oba projekty DrvTestCPP i DrvTestCS jsou k dispozici ve zdrojové formě
- Pro převod byla použita technika P-Invoke, potřebné funkce Win32 API byly importovány pomocí atributu DllImport
- Příklady v obou projektech jsou realizovány následujícími funkcemi:
  - **EjectMedia()** – otevře mechaniku CD/DVD
    - Používá **IOCTL_STORAGE_EJECT_MEDIA**
  - **LoadMedia()** – zavře mechaniku CD/DVD
    - Používá **IOCTL_STORAGE_LOAD_MEDIA**
  - **GetDriveGeometryEx()** – zjistí informace o fyzickém disku
    - Používá **IOCTL_DISK_GET_DRIVE_GEOMETRY_EX**
  - **GetPartitionInfoEx()** – zjistí informace o vybrané partition
    - Používá **IOCTL_DISK_GET_PARTITION_INFO_EX**
- Pozor při vlastních pokusech !!! Mezi IOCTL kódy existují i takové na zápis do partition tabulky nebo formátování disku.

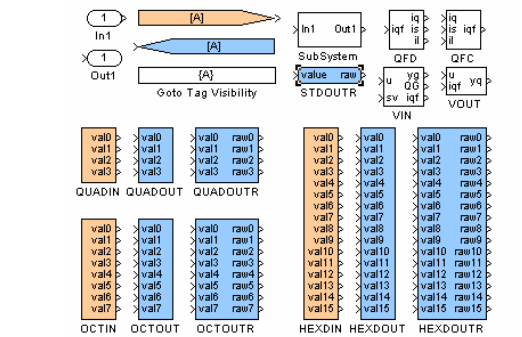
Architektrura ŘS REX



Ovladače a moduly ŘS REX

- Vstupně-výstupní ovladače slouží pro připojení vstupů a výstupů reálných procesů do ŘS REX prostřednictvím tzv. vstupně-výstupního subsystému (I/O subsystem)
- Ovladače v ŘS REX jsou implementovány v tzv. modulech, které na platformě Windows, Windows CE a Phar Lap ETS mají formu DLL knihoven.
- Pro každý modul **<Modul>** existují 2 dll knihovny:
  - **<Modul>_H.dll** – vývojová (host) část modulu. Je používána pro:
    - Konfiguraci ovladače z modulu v programech RexDraw a Simulink
    - Překlad konfigurace v programu RexComp
  - **<Modul>_T.dll** – cílová (target) část modulu a jeho ovladače
    - Slouží pro zprostředkování vstupně výstupních operací v programu RexCore
- Každý ovladač je implementován třídou v jazyku C++, odvozenou od základní třídy XIODriver. Daný modul může implementovat několik ovladačů

Knihovna InOutLib



Vstupní bloky

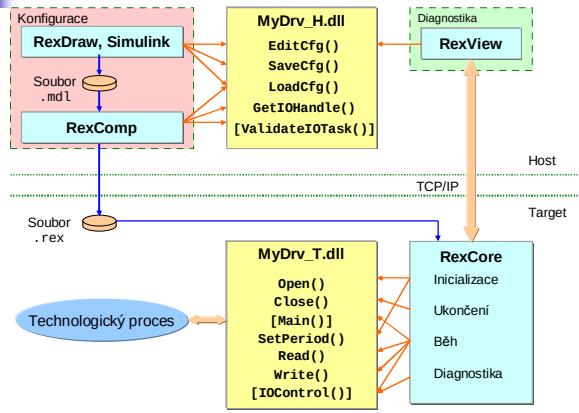
Výstupní bloky

Rozhraní modulů ŘS REX

Každý modul systému REX zveřejňuje dvě globální funkce:

- **GetModuleVersion()**
  - Vrací verzi modulu, která je porovnána s verzí ŘS REX
  - Pokud jsou obě verze navzájem nekompatibilní vrací se chyba
  - Funkce je volána ihned po zavedení modulu do paměti. V případě vrácení chyby, je modul z paměti uvolněn a tato chyba je vrácena jako chyba systému REX
- **RegisterModule()**
  - Volána po úspěšném zavedení modulu do paměti a úspěšném zavolání **GetModuleVersion()**
  - Registruje do ŘS všechny třídy, které mohou být od tohoto okamžiku nadále používány
  - V případě modulu ovladačů jsou zaregistrovány všechny ovladače
- Dále předpokládejme nejjednodušší situaci, kdy daný modul obsahuje právě jeden ovladač

Nejdůležitější funkce ovladače ŘS REX



Základní typy ovladačů ŘS REX

- Jednoduchý ovladač bez vlastní úlohy OS (threadu)
  - Vhodný pro přímo připojená zařízení, z/do nichž lze přečíst/nastavit hodnoty velmi rychle (v řádu mikrosekund)
  - Čtení vstupů a nastavování výstupů se provádí na kontextu úloh ŘS REX
- Ovladač s vlastní úlohou OS
  - Vhodný při větší časové náročnosti čtení/nastavování hodnot z/do zařízení, např. pro připojení pomocí komunikace (např. sériová linka)
  - Pak čtení/zápis probíhá v samostatné úloze (threadu) OS, asynchronně s během úloh ŘS REX
  - Vzájemná výměna dat je přes sdílenou pamět (cache). Musí se používat synchronizační objekty (mutexy, semafory)
- Ovladač spouštějící úlohy ŘS REX
  - Nejsložitější typ ovladače, vhodný pro speciální účely, např. pro spouštění velmi rychlých úloh od externího přerušení (interruptu)
  - Takové úlohy se do konfigurace exekutivy zařazují pomocí bloků IOTASK připojovaných k ovladačům konfigurovaným pomocí bloků TIODRV
  - Tento typ lze kombinovat s předchozím typem

Nejdůležitější metody ve vývojovém prostředí

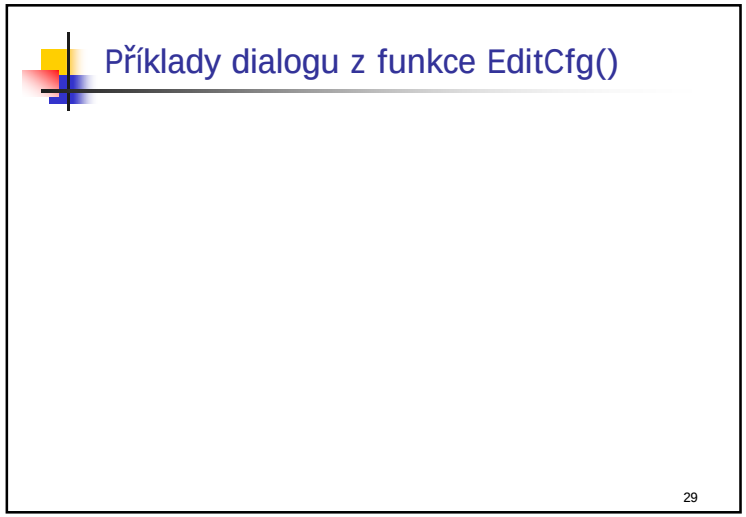
- **`EditCfg()`** – konfigurace vlastního ovladače volaná z RexDraw nebo Simulinku
  - Obvykle implementuje konfigurační dialogové okno
  - Pro ovladače s pevnými jmény vstupů a výstupů může být prázdná
- **`GetIOHandle()`** – funkce pro získání handlu daného vstupního/výstupního signálu.
  - Je volána pro všechny vstupně výstupní bloky z knihovny RexLib/InOutLib
  - Dále nastavuje typ každého signálu (např. **`XBOOL`**, **`XLONG`**, **`XDOUBLE`**)
- **`SaveCfg()`** – ukládá konfiguraci vytvořenou pomocí **`EditCfg()`** do souboru (s příponou **`.rio`** – REX I/O) na disk (např. v textovém formátu).
  - Jméno souboru se zadává jako parametr bloků **`IODRV`** a **`TIODRV`**
- **`LoadCfg()`** – načte konfiguraci ze souboru uloženého funkcí **`SaveCfg()`**
  - Je volána před funkcí **`EditCfg()`**. Pokud vrátí chybu, je funkce EditCfg() volána jen pokud si uživatel přeje vytvořit nový konfigurační soubor
- **`GetIODrvStatus()`** – vrací textový řetězec odpovídající číselnému kódu stavu ovladače (obvykle chyby)
  - Je volána z RexView pro výpis stavu ovladače

Nejdůležitější metody v cílovém prostředí (1/2)

- **`Open()`** – otvírá (inicializuje) ovladač
  - Volána při inicializaci RexCore dřív než inicializace řídicích úloh
  - Může navázat spojení se zařízením, alokovat paměť, inicializovat výstupy, apod.
- **`Close()`** – zavírá (ukončuje) činnost ovladače, opačná funkce než **`Open()`**
  - Volána při ukončování běhu RexCore později než ukončovací funkce řídicích úloh
  - Může např. uvolnit paměť nastavit výstupy na bezpečné hodnoty, apod.
- **`Main()`** – hlavní funkce ovladače s vlastní úlohou OS
  - Je periodicky volána exekutivou reálného času
  - Slouží např. pro vlastní komunikaci vstupů a výstupů s cílovým zařízením
  - Pro ovladače bez vlastní úlohy OS se neimplementuje
- **`SetPeriod()`** – nastavení periody spouštění každého vstupního a výstupního bloku do ovladače
  - Ovladač může získanou informaci o periodě vzorkování/aktualizace daného vstupu/výstupu použít k optimalizaci komunikace s příslušným zařízením

Nejdůležitější metody v cílovém prostředí (2/2)

- **`Read()`** – čte vstupní signály z ovladače
  - Volána ze vstupních bloků knihovny InOutLib
  - V případě ovladače bez vlastní úlohy OS musí funkce přečíst vstupy z daného zařízení, jinak získává hodnoty za vyrovnávací paměti cache
- **`Write()`** – nastavuje výstupní signály z ovladače
  - Volána z výstupních bloků knihovny InOutLib
  - V případě ovladače bez vlastní úlohy OS musí funkce nastavit výstupy do daného zařízení, jinak nastavuje hodnoty výstupů do vyrovnávací paměti cache
  - Pro bloky **`STDOUTR`**, **`QUADOUTR`**, **`OCTOUTR`** a **`HEXDOUTR`**, může nastavovat do jejich výstupů **`raw`** nebo **`raw<i>`** výsledky zápisu na fyzické zařízení, včetně příznaků kvality signálu (jako v OPC)
- **`IOControl()`** – speciální funkce ovladače pro účely, které není možné zařídit jinou funkcí
  - Myšlenka použití funkce je podobná jako u **`DeviceIoControl()`** z Win32
  - Funkci lze vzdáleně volat z diagnostického programu DDDShell systému REX a v blízké budoucnosti ji bude možno volat i z programu RexView.
  - Parametry této funkce lze konfigurovat ve vývojovém prostředí ve funkci **`EditCfg()`**


Příklady dialogu z funkce EditCfg()
29