P-IRIS Controller

C++ SDK

Návod k použití

atesystem
FOCUSED ON DETAIL

ATEsystem s.r.o.
Studentská 6202/17
708 00 Ostrava-Poruba
Česká republika

M +420 595 172 720
E produkty@atesystem.cz
W www.atesystem.cz

INFORMACE O DOKUMENTU

Č. revize	Autor	Datum revize	Popis
1	Pařez Jakub	16.01.2019	Vytvoření dokumentu
2	Navrátil Josef	04.04.2019	Korektura
3	Pařez Jakub	25.04.2019	Transformování obrázků do textu

Přílohy

Poznámky

Kontakt

ATEsystem s.r.o.
Studentská 6202/17
708 00 Ostrava 8 – Poruba
Česká republika

T +420 595 172 720
F +420 595 170 100
E produkty@atesystem.cz
W www.atesystem.cz

Všechna práva vyhrazena. Žádná část tohoto dokumentu nesmí být publikována, přenášena na jakémkoliv médiu, kopírována ani překládána do cizích jazyků bez předchozího písemného souhlasu firmy ATEsystem s.r.o.

ATEsystem s.r.o. nepřejímá žádné záruky za obsah tohoto dokumentu a případné tiskové chyby.

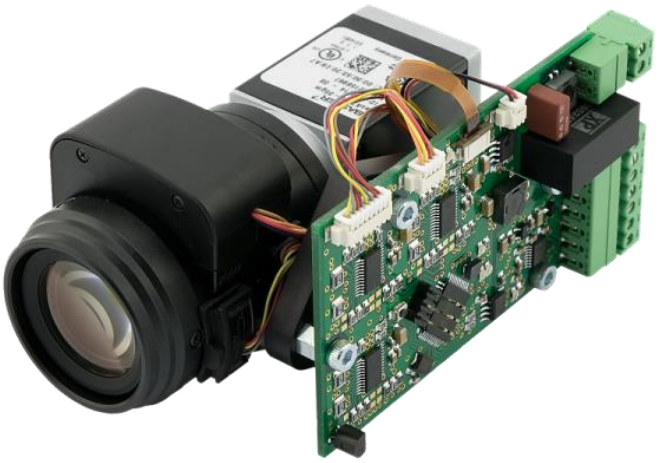
V dokumentu jsou použité názvy produktů, firem, které mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

ATEsystem s.r.o. © 2019

OBSAH

1 POPIS

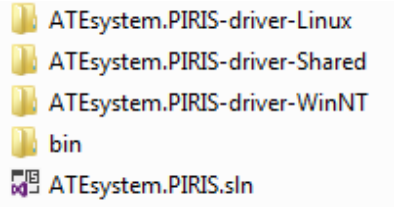
ATEsystem.PIRIS-driver je součást softwarové podpory pro produkt P-IRIS Controller. Slouží jako hlavní rozhraní mezi hardwarem a nadřazenou ovládací aplikací. Umožňuje kompletní ovládání jednotky ve verzi **UART/Ethernet** i ve verzi **RS232**. Implementuje proprietární komunikační ATEsystem protokol, který je popsán v katalogovém listu produktu. Ovladač je napsán v C++17 a je distribuován formou zdrojového kódu pod licencí MIT. Závislosti tohoto softwaru jsou **Pylon 5.0.12** a v případě OS Windows balíčky **VC++ Redistributable 2013** a **2017**. Ovladač používá open-source software **Serial**. Závislosti jsou distribuovány ve složkách *bin* a *install* a jsou nezbytné pro správnou funkčnost. Ovladač včetně závislostí je multiplatformní, v distribuci lze nalézt spustitelná dema jak pro OS Windows (x86, amd64) tak pro OS Linux (x86, amd64, armhf, arm64). K vývoji bylo použito IDE Microsoft Visual Studio 2017, které umožňuje multiplatformní vývoj C++ aplikací, příslušné nastavení IDE a další detaily jsou popsány níže.



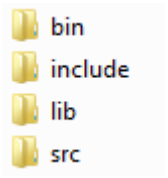
Obr. 1 – Řídicí jednotka pro objektiv s krokovými motory

2 INSTALACE

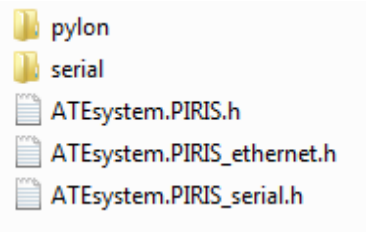
Po rozbalení distribuovaného archívu naleznete SLN soubor, který slouží k otevření celé *Solution* v IDE Visual Studio 2017. Ve složkách **Linux** a **WinNT** jsou VS projekty s příslušným nastavením kompileru a linkeru zvlášť pro architektury x86, amd64 a armhf a arm64. Složka **Shared** je hlavní kontejner obsahující zdrojový kód ovladače, hlavičkové soubory, knihovny a příklady. Ve složce **bin** jsou spustitelné soubory příkladů.



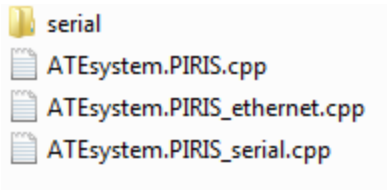
Zdrojový kód ovladače je rozčleněn do čtyř složek bin, include, lib a src. Ve složce **bin** jsou binární závislosti zvlášť pro každou architekturu. Jedná se o dynamické knihovny Pylon a GenICam, které je nutné libovolným způsobem nainstalovat do cílového stroje. V případě OS Windows je v adresáři připraven dávkový soubor, nicméně pokud si uživatel není jistý, je v pořádku nainstalovat Pylon 5 runtime z oficiálního zdroje. Ve složce **lib** jsou staticky linkované knihovny pro OS Windows.



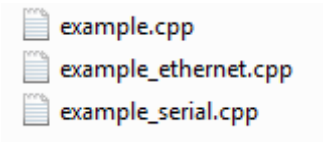
Ve složce **include** jsou hlavičkové soubory ovladače, ATEsystem.PIRIS je hlavní a jediný hlavičkový soubor, který je třeba includovat do volající aplikace. Dále ve složkách **pylon** a **serial** jsou příslušné hlavičky pro UART/Ethernet, resp. RS232 verzi jednotky.



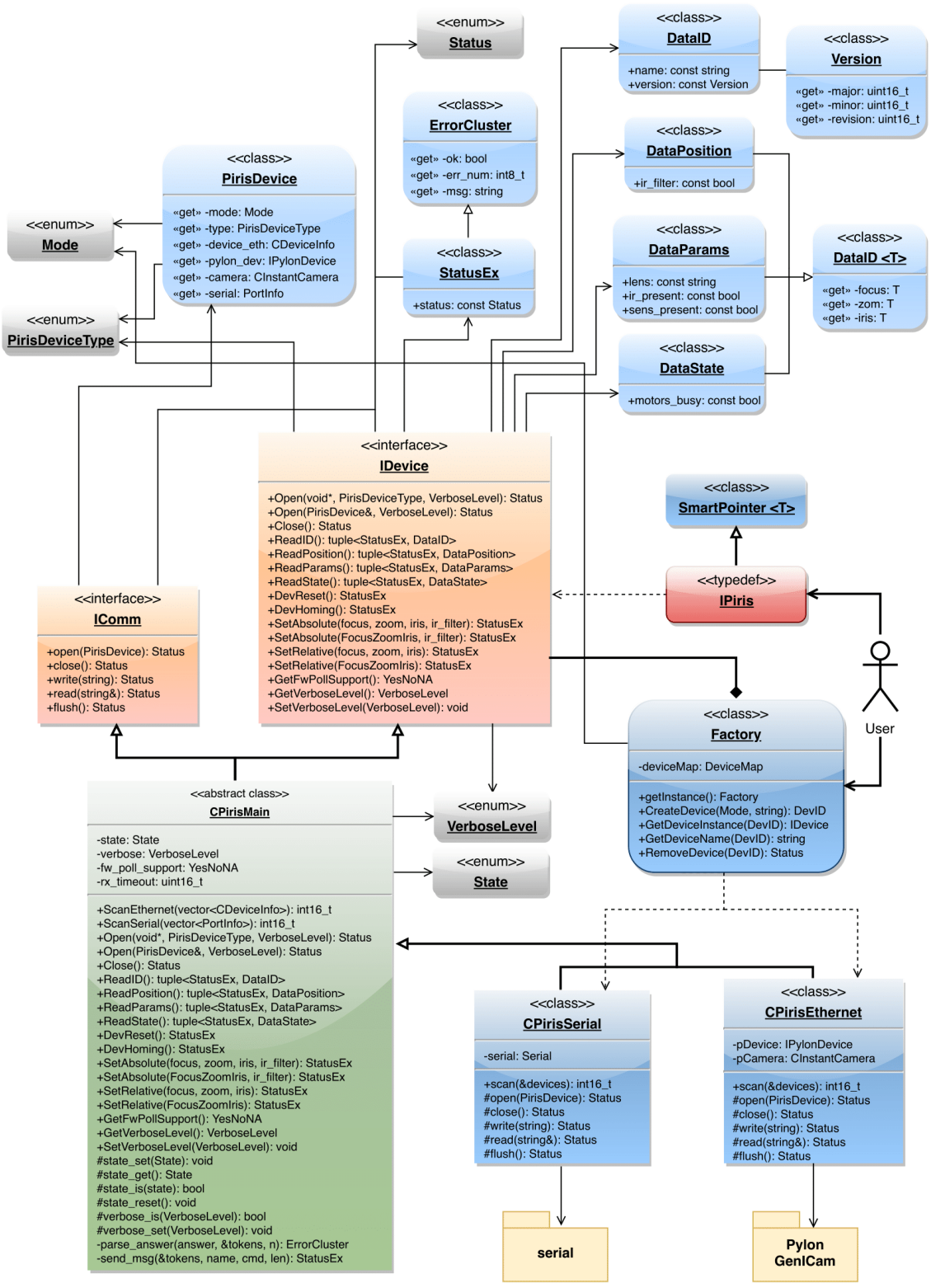
Ve složce **src** jsou veškeré zdrojové kódy. Přímo v kořenu složky jsou kódy ovladače, ve složce **serial** jsou zdrojové kódy knihovny pro multiplatformní manipulaci se sériovými porty od Williama Woodalla a Johna Harrisona.



Dále jsou v kořenové složce ovladače obsaženy ukázky použití ve formě demo souborů, pojmenovaných **example**. Soubor *example.cpp* je hlavní vstupní demo soubor, který se dále větví do souborů se sufixem *_ethernet* a *_serial*, podle zvoleného režimu funkce. Odpovídající binární spustitelné soubory tohoto dema lze nalézt ve složce **/bin** v kořenovém adresáři ovladače.



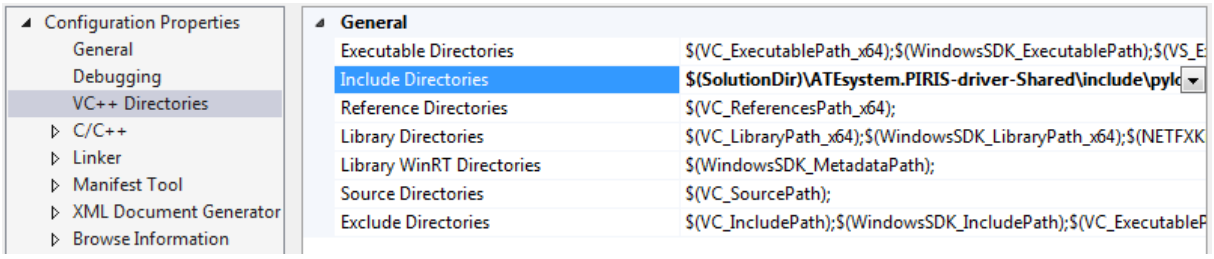
3 TŘÍDNÍ DIAGRAM



4 NASTAVENÍ IDE

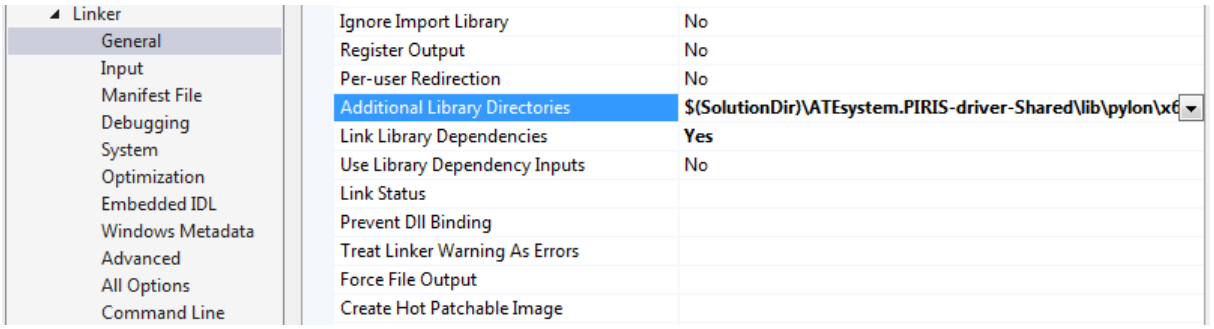
Pro vytvoření spustitelného souboru ze zdrojového kódu ovladače je nezbytné správně nastavit *build* nástroje. Nastavení kompileru, linkeru a dalších důležitých parametrů bude popsáno na vývojovém prostředí Microsoft Visual Studio 2017 Enterprise.

V nastavení projektu v *Configuration Properties* > *VC++ Directories* se nastaví **Include Directories**.



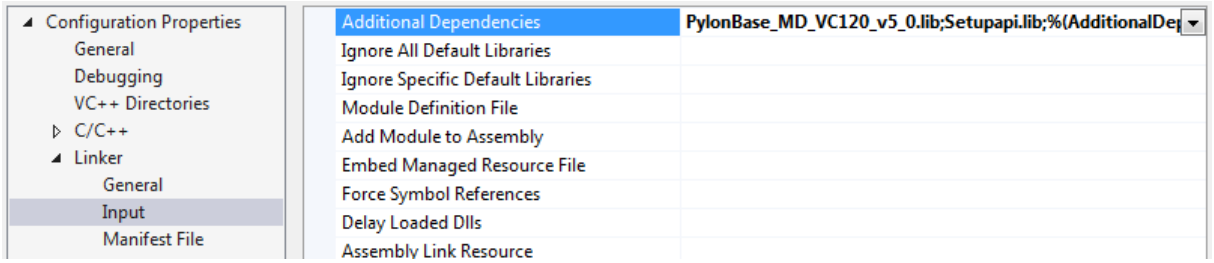
1) $(SolutionDir)\ATEsystem.PIRIS-driver-Shared\include\pylon\WinNT
2) $(SolutionDir)\ATEsystem.PIRIS-driver-Shared\include\serial
3) $(SolutionDir)\ATEsystem.PIRIS-driver-Shared\include

V nastavení projektu v *Configuration Properties* > *Linker* > *General* se nastaví **Additional Library Directories** na cestu statické Pylon knihovny ze složky lib/pylon a příslušné architektury.



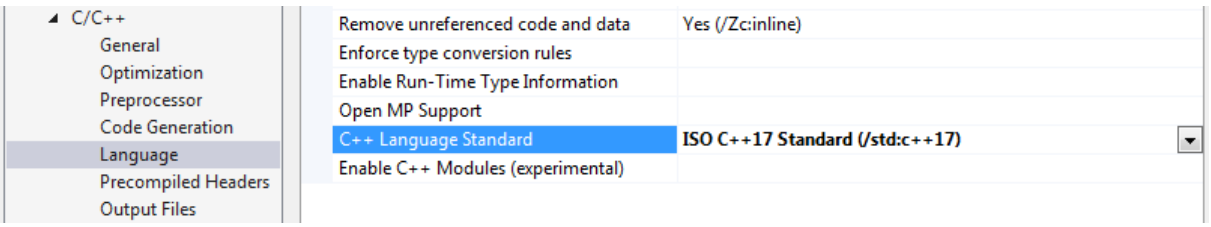
1) $(SolutionDir)\ATEsystem.PIRIS-driver-Shared\lib\pylon\Win32

V nastavení projektu v *Configuration Properties* > *Linker* > *Input* se nastaví **Additional Dependencies** na název statické Pylon knihovny a v případě OS Windows také na název knihovny *Setupapi*.



1) PylonBase_MD_VC120_v5_0.lib
2) Setupapi.lib

V nastavení projektu v *Configuration Properties* > *C/C++* se nastaví **Language** na *ISO C++17 Standard*.



- ISO C++17 Standard (/std:c++17)

5 POUŽITÍ

Pro hlavní použití ovladače jsou zásadní dva objekty, třída `Factory` (Singleton) a rozhraní `IPiris`, které je ve skutečnosti Smart Pointer. Dále pak objekty sloužící jako datové kontejnery (`DataID`, `DataPosition`, `DataParams` a `DataState`), vráceny z funkcí přes hodnotu pomocí kolekce `std::tuple` a objekt `StatusEx`, nesoucí základní informace o provedené akci, včetně chybového kódu, chybové zprávy zařízení – `ErrorCluster` a chybové zprávy ovladače – `Status`.

`Factory` je třída implementována dle návrhového vzoru factory. Slouží k vytváření instancí ovladače pro konkrétní komunikační rozhraní (UART/Ethernet – `CPirisEthernet`, RS232 – `CPirisSerial`). Takových instancí lze vytvořit více a pracovat s nimi paralelně, pomocí jediné instance factory, jako identifikátor slouží `DevID_t`. Pomocí identifikátoru lze instance vytvářet, mazat a lze přistupovat k jejím rozhraním. Jelikož je toto rozhraní Smart Pointer, není třeba na něj volat `delete`, ani používat funkci *`RemoveDevice()`*, ta je vhodná pro správu více než jedné instance ovladače. Díky tomuto systému lze transparentně a bez rizik „memory leaku“ používat klidně několik jednotek P-IRIS Controller najednou.

```
DevID_t CreateDevice(Mode mode = Mode::ETHERNET);
DevID_t CreateDevice(const std::string& name, Mode mode);
IDevice* GetDeviceInstance(DevID_t id);
std::string GetDeviceName(DevID_t id);
Status RemoveDevice(DevID_t id);
```

`IPiris` je hlavní rozhraní pro manipulaci s instancí ovladače. Ve skutečnosti se jedná o Smart Pointer typu IDevice `typedef SmartPointer<IDevice> IPiris`. Funkce tohoto rozhraní jsou popsány v tabulce níže. Pro nastavování hodnot je zásadní funkce `SetAbsolute()`, která lze zavolat buď přímo s numerickými hodnotami pro ostření, zoom, clonu a IR filtr nebo s třídou `FocusZoomIris<T>`, která tyto hodnoty obaluje do generického kontejneru. Tento kontejner také slouží pro vrácení hodnot a stavových informací z funkcí `ReadPosition()`, `ReadParams()`, `ReadState()`, proto je vhodné v některých případech použít první způsob a v jiných tento kontejner. V některých případech je také možné použít funkci, která nenastavuje absolutní hodnoty, ale relativní, `SetRelative()`.

Je nezbytné zavolat ***`ReadID()`*** ihned po ***`Open()`***, kvůli správné detekci kompatibilní verze firmwaru.

```
Interface IDevice
{
public:

    virtual Status Open(void* dev, PirisDeviceType type, VerboseLevel verbose) = 0;
    virtual Status Open(const PirisDevice& dev, VerboseLevel verbose) = 0;
    virtual Status Close() = 0;

    virtual std::tuple<StatusEx, DataID> ReadID() = 0;
    virtual std::tuple<StatusEx, DataPosition> ReadPosition() = 0;
    virtual std::tuple<StatusEx, DataParams> ReadParams() = 0;
    virtual std::tuple<StatusEx, DataState> ReadState() = 0;

    virtual StatusEx DevReset() = 0;
    virtual StatusEx DevHoming() = 0;

    virtual StatusEx SetAbsolute(uint16_t focus = 0,
                                 uint16_t zoom = 0,
                                 uint16_t iris = 0,
                                 bool ir_filter = false) = 0;
    virtual StatusEx SetAbsolute(const FocusZoomIris<uint16_t>& values,
                                 bool ir_filter = false) = 0;
    virtual StatusEx SetRelative(int16_t focus = 0,
                                 int16_t zoom = 0,
                                 int16_t iris = 0) = 0;
    virtual StatusEx SetRelative(const FocusZoomIris<int16_t>& values) = 0;

    virtual YesNoNA GetFwPollSupport() = 0;
    virtual VerboseLevel GetVerboseLevel() = 0;
    virtual void SetVerboseLevel(VerboseLevel level) = 0;

    virtual ~IDevice() = 0;
};
```

Návratová hodnota	Název	Parametry	Popis
Status	Open	void* dev PirisDeviceType type VerboseLevel verbose	Otevře spojení se zařízením pomocí ukazatele a následným přetypováním. Vhodné pro implementaci knihovny nebo volání funkce z jiného prostředí.
Status	Open	const PirisDevice& dev VerboseLevel verbose	Otevře spojení se zařízením pomocí třídy *PirisDevice*, která se vytvoří z dat získaných z vyhledávacích funkcí.
Status	Close		Uzavře spojení se zařízením.
tuple<StatusEx, DataID>	ReadID		Pošle příkaz **IDN** a vrátí objekt *StatusEx* a *DataID*, ve kterém je *name*: název jednotky a *version*: verze firmware
tuple<StatusEx, DataPosition>	ReadPosition		Pošle příkaz **GP** a vrátí objekt *StatusEx* a *DataPosition*, ve kterém je *position*: aktuální pozice ostření, zoom a clony a *ir_filter*: IR filtr je zapnutý.

tuple<StatusEx, DataParams>	ReadParams		Pošle příkaz **GT** a vrátí objekt *StatusEx* a *DataParams*, ve kterém je *max_value*: maximální hodnoty pozice ostření, zoom a clony, *lens*: typ objektivu, *ir_present*: IR filtr je přítomen a *sens_present*: koncový snímač přítomen.
tuple<StatusEx, DataState>	ReadState		Pošle příkaz **GS** a vrátí objekt *StatusEx* a *DataState*, ve kterém je *state*: stav motorů (OK/ERR) a *motors_busy*: motory jsou právě v pohybu.
StatusEx	DevReset		Pošle příkaz **RST.** Tím se iniciuje restart celého zařízení. Je nutno počkat minimálně 10 sekund.
StatusEx	DevHoming		Pošle příkaz **INI.** Nastavení ostření a zoomu na inicializační polohu, nastavení clony na plně otevřenou, zapnutí IR filtru.
StatusEx	SetAbsolute	uint16_t focus uint16_t zoom uint16_t iris bool ir_filter	Pošle příkaz **SETA**:FX;ZX;PX;IX, který nastaví ostření, zoom a clonu do absolutních pozic specifikovaných v parametrech a zapne nebo vypne IR filtr.
StatusEx	SetAbsolute	const FocusZoomIris<uint16_t>& values bool ir_filter	Pošle příkaz **SETA**:FX;ZX;PX;IX, který nastaví ostření, zoom a clonu do absolutních pozic specifikovaných v parametrech a zapne nebo vypne IR filter.
StatusEx	SetRelative	int16_t focus int16_t zoom int16_t iris	Pošle příkaz **SETR**:FX;ZX;PX, který posune ostření, zoom a clonu o relativní hodnoty specifikované v parametrech.
StatusEx	SetRelative	const FocusZoomIris<int16_t>& values	Pošle příkaz **SETR**:FX;ZX;PX, který posune ostření, zoom a clonu o relativní hodnoty specifikované v parametrech.
YesNoNA	GetFwPollSupport		Vrátí informaci ohledně dostupnosti cyklického vyčítání statusu (verze FW 1.7.2 a výše).
VerboseLevel	GetVerboseLevel		Vrátí aktuálně nastavenou úroveň výpisu informací na standardní výstup.
void	SetVerboseLevel	VerboseLevel level	Nastaví úroveň výpisu informací na standardní výstup.

Tab. 1 – Popis funkcí hlavního rozhraní ovladače

Vyhledávání zařízení je řešeno pomocí dvou statických funkcí, které vrací vektor příslušných objektů – deskriptorů podle typu hlavní instance. `Pylon::CDeviceInfo` je deskriptor UART/Ethernetových zařízení, `serial::PortInfo` je deskriptor RS232 zařízení. Tyto objekty slouží pro vytvoření instance třídy `PirisDevice`, která popisuje vždy pouze jedno zařízení – produkt, a která slouží jako hlavní vstupní objekt funkce *`Open(const PirisDevice& dev)`*.

```
static int16_t ScanEthernet(std::vector<Pylon::CDeviceInfo>& devices, bool verbose);
static int16_t ScanSerial(std::vector<serial::PortInfo>& devices, bool verbose);
```

Návratová hodnota	Název	Parametry	Popis
int16_t	ScanEthernet	vector<Pylon::CDeviceInfo>& devices bool verbose	Vyhledá všechny dostupné zařízení ve verzi UART/Ethernet, vrátí jejich seznam ve vektoru *devices* a jejich počet návratovou hodnotou. (-1: chyba)
int16_t	ScanSerial	std::vector<serial::PortInfo>& devices bool verbose	Vyhledá všechny dostupné zařízení ve verzi RS232, vrátí jejich seznam ve vektoru *devices* a jejich počet návratovou hodnotou. (-1: chyba)

Tab. 2 – Popis statických funkcí pro vyhledávání

Poznámka: Do verze firmwaru **1.5.2** včetně nebylo možné kontinuálně vyčítat stav během provádění jakékoliv akce (odeslal se příkaz, začala se provádět akce, a po skončení akce se vrátila zpráva o provedení – „OK\r\n“). Není tedy dopředu známé, jak dlouho se musí čekat. Od následujícího vydání firmwaru **1.7.2** je toto opraveno. Pošle se příkaz, ihned se vrátí odpověď a začne se provádět akce. Během průběhu akce je vhodné se cyklicky dotazovat na stav (GS) a kontrolovat stav *motors_busy*, který bude během provádění nastaven na *True*, po skončení se pak nastaví na *False*.

6 PŘÍKLADY

Ve složce *Shared* jsou zdrojové kódy funkčních příkladů použití nazvané *example*, demonstrující veškeré možnosti ovladače jak s UART/Ethernet, tak s RS232 verzí jednotky. Ve složce **bin** jsou odpovídající spustitelné soubory zvlášť pro OS **Linux** (x86, x64, ARM, ARM64) a OS Windows (x86, x64). V případě OS **Windows** je třeba mít nainstalované balíčky VC++ Redistributable 2013 a 2017 ze složky **install**. Pokud na cílovém stroji není nainstalovaný Pylon 5 runtime, je třeba jej především v případě Linuxu nainstalovat. Lze použít oficiální soubory z webové stránky www.baslerweb.com nebo využít přiložené dynamické knihovny, zvlášť pro určitý OS a pro konkrétní CPU architekturu. Pokud je použit OS Windows, jsou automaticky preferovány dynamické knihovny, které jsou přiloženy ve stejném adresáři.

OS Windows nabízí pro UART/Ethernetovou verzi **Pylon GUI**, kdy po spuštění dema a zvolení Ethernet režimu dojde k otevření okna s živým náhledem kamery. Pokud z nějakého důvodu toto GUI nefunguje nebo není vyžadováno, lze demo spustit s parametrem **--no-gui**. Nastavení parametrů kamery je třeba udělat před spuštěním dema, nebo využít přednastavený výchozí konfigurační soubor **acA2040-35gmATE.pfs** pro kameru Basler. Pokud je tento soubor ve stejném adresáři jako demo, je uživatel vyzván k potvrzení nahrání této konfigurace do kamery.

Po spuštění programu si uživatel vybere verzi jednotky dle typu komunikačního rozhraní, **Serial** pro RS232 a **Ethernet** pro verzi UART/Ethernet.

```
This is ATEsystem P-IRIS Controller driver example.
www.atesystem.cz  produkty@atesystem.cz
================================================

Choose device connection mode:
  0 : Serial
  1 : Ethernet
> 1
```

Dále je vybrán režim dema. **User input** znamená režim, kdy je uživatel manuálně nastavuje parametry, **Sequence** je režim, kdy se sekvenčně projedou postupně všechny parametry od nuly po maximum a zpět, ostření, zoom, clona, IR filtr a na závěr všechno naráz.

```
Choose demo mode:
  0 : User input
  1 : Sequence
> 1
```

Pokračuje se výběrem ze seznamu připojených zařízení. V případě UART/Ethernet verze je nutné nejprve správně nakonfigurovat kameru Basler např. pomocí Pylon IP Configuratoru. Ke kameře nesmí být nikdo připojen a musí být ve stejné podsíti jako PC.

```
Choose device number:
  0. Basler acA2040-35gmATE#[illegible]
> 0
```

Pokud je vybrána RS232 verze jednotky, je možné použít libovolný RS232/USB převodník.

```
Choose device number:
  0. COM15   USB Serial Port (COM15)
  1. COM13   Standard Serial over Bluetooth link (COM13)
  2. COM6    Standard Serial over Bluetooth link (COM6)
  3. COM14   Standard Serial over Bluetooth link (COM14)
  4. COM1    Lantronix CPR Port (COM1)
  5. COM7    Standard Serial over Bluetooth link (COM7)
>
```

V případě UART/Ethernet verze je dále uživatel vyzván k výběru konfigurace kamery, pokud ve složce example existuje konfigurační soubor s příponou PFS. Možnost **Load default** načte výchozí nastavení do kamery ze souboru, přepíše tedy veškerou předchozí konfiguraci. **Use current** použije stávající.

```
PIRIS open - success (Ethernet camera (camera overiden))
Camera Device Information
=========================
Vendor           : Basler
Model            : acA2040-35gmATE
Firmware version : 107603-01;U;acA2040_35gmATE;V1.0-2;0
=========================
PIRIS set up - success


Choose camera configuration:
  0 : Load default
  1 : Use current
>
```

Následuje zahájení komunikace s jednotkou. Vyčte se název zařízení, typ objektivu, verze firmwaru, aktuální a maximální pozice ostření, zoomu, clony a IR filtru, přítomnost koncových spínačů, indikace běhu motorů, chybová zpráva zařízení a chybová zpráva ovladače. Pak proběhne inicializace objektivu do výchozího stavu.

```
Camera configuration file acA2040-35gmATE.pfs successfully loaded.

CMD_READ_ID     success:  name=           P_IRIS
                          version=        1.7.2

CMD_READ_POS    success:  focus=          2025
                          zoom=           0
                          iris=           0
                          ir_filter=      1

CMD_READ_TYPE   success:  type=           TL1250P
                          focus=          2025
                          zoom=           800
                          iris=           19
                          ir_present=     1
                          sens_present=   1

CMD_READ_STATE  success:  focus=          1
                          zoom=           1
                          iris=           1
                          busy=           0

CMD_HOMING      success

Wait for action finish....
```

V režimu **User Input** se program uživatele cyklicky dotazuje na zadání čtyř hodnot v rozsahu v hranatých závorkách, ostření, zoom, clona a IR filtr.

```
(enter any string to quit)
The User Input Demo will start now.Press any key to continue . . .

------------------------------------------------------------
Enter Focus <0;2025> Zoom <0;800> Iris <0;19> and IR <0;1> in format: 0 0 0 0:
> 2000 250 3 1
```

V režimu **Sequence** program automaticky nastavuje hodnoty ostření, zoomu, clony a IR filtru v následujícím pořadí: Ostření min. -> ostření max. -> ostření min., zoom min. -> zoom max. -> zoom min., clona min. -> clona max. -> clona min., IR filtr vypnut -> IR filtr zapnut -> IR filtr vypnut, vše min. -> vše max.

```
The Demo Sequence will start now. Press any key to continue . . .

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

Posílání příkazů do řídící jednotky je v obou režimech realizováno stejným způsobem. Nejprve se pošle příkaz, kterým se nastaví objektiv na absolutní hodnoty. Pak se čeká na dokončení akce, v závislosti na verzi firmwaru různým způsobem (popsáno v kapitole 5 v poznámce na konci). Následuje vyčtení pozice pro ověření.

```
CMD_SET_ABS     success:  focus=          2000
                          zoom=           250
                          iris=           3
                          ir_filter=      1

Wait for action finish....

CMD_READ_POS    success:  focus=          2000
                          zoom=           250
                          iris=           3
                          ir_filter=      1
```

Program se ukončí v režimu Sequence automaticky po cca 45 sekundách. V režimu User Input stačí pro ukončení napsat a potvrdit jakýkoli znakový řetězec. Před uzavřením se vždy provede restart jednotky a až pak následuje uzavření spojení.

```
Enter Focus <0;2025> Zoom <0;800> Iris <0;19> and IR <0;1> in format: 0 0 0 0:
> exit

------------------------ DEMO QUIT -------------------------
CMD_RESET       success


Wait for action finish and then Press any key to continue . . .

PIRIS close - success

Press any key to continue . . .
```