VirtualLoop

Manual do Usuário

Biblioteca de Software para detecção de objetos por Análise de Vídeo

Release: v4.6.0
Data: 15/12/2020

Sumário

Histórico de alterações

Data	Versão	Revisão
29/06/2017	2.3.8	• Versão Inicial do documento
05/10/2017	2.4.0	• Atualização da documentação da API para a versão 2.4.0 do SDK • Adicionado suporte a múltiplas regiões, velocidade e objetos abandonados
14/06/2018	2.8.4	• Correção do funcionamento das threads
16/01/2019	3.5.0	• Adição de wrapper Java
18/04/2019	3.6.0	• Adição da funcionalidade de classificação de tipos de veículos • Adição da API para requisitar informações sobre a licença
29/07/2019	3.7.0	• Adição de APIs para mudar e requisitar valores de configurações do tipo float • Correção no wrapper Java • Classificação veicular desativada por padrão
05/06/2019	4.0.0	• Alteração no nome da biblioteca, includes, e prefixo de códigos de retorno
13/08/2019	4.1.0	• Otimizações para reduzir tempo de processamento
09/10/2019	4.2.0	• Atualização do modelo de detecção de veículos • Classificação veicular ativada por padrão • Otimizações para reduzir tempo de processamento e correções
29/11/2019	4.3.0	• Contagem de motos desativada • Melhorias na lógica de ativação das regiões ativas • Correção de vazamento de memória do wrapper Java
01/12/2019	4.4.0	• Contagem regular de veículos • Rastreamento de objetos atulizado e ativado por padrão
26/12/2019	4.5.0	• Adição da funcionalidade de classificação veicular ao Windows • Atualização do wrapper Python
20/03/2020	4.5.1	• Correção do bug de não geração de eventos na Vigia-VL • Atualização da estrutura de carga de modelos
22/04/2020	4.5.2	• Correcão de comportamento para o caso de a ITSCAM não possuir campo TempoCaptura • Correção do update dos objetos • Correção da degradação do modelo de fundo • Adição de API para não ativar o laço com determinados tipos de veículo • Atualização do modelo de detecção de veículo
08/06/2020	4.5.3	• Correção de comportamento para diminuir casos de dupla ativação em veículos longos • Correção dos wrappers incluindo defines faltantes • Atualização do mínimo hardware necessário
18/08/2020	4.5.4	• Adição do modo de operação para produtos de ocupação • Atualização na documentação
15/12/2020	4.6.0	• Suporte a contagens múltiplas por ativação • Controle pela API do tempo máximo de ativação • Melhoria geral do algoritmo de tracking • Adição do wrapper C# • Atualização dos wrappers Python e Delphi • Atualização da documentação

1. Prefácio

1.1. Condições Gerais

Os dados e as informações contidas neste documento não podem ser alterados sem a permissão expressa por escrito da Pumatronix Equipamentos Eletrônicos. Nenhuma parte deste documento pode ser reproduzida ou transmitida para qualquer finalidade, seja por meio eletrônico ou físico.

1.2. Licença de software

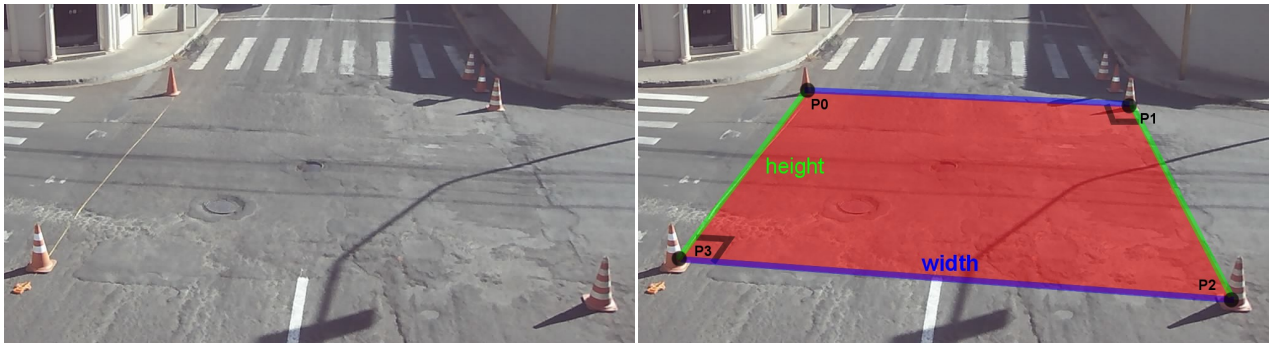
O software e a documentação em anexo estão protegidos por direitos autorais. Ao instalar o software, você concorda com as condições do contrato de licença.

2. Introdução

VirtualLoop é uma biblioteca de software desenvolvida pela **Pumatronix Equipamentos Eletrônicos** especializada na implementação de laços virtuais através de análise de vídeo. Um laço virtual é uma região da imagem onde se deseja detectar a presença de um objeto. Esta área é delimitada por qualquer conjunto de quatro pontos não colineares (qualquer quadrilátero convexo), limitado ao tamanho da imagem. O **VirtualLoop** define 4 tipos diferentes de laços virtuais, cada uma servindo a um propósito diferente. Ao longo desse manual os termos laços virtuais e regiões referem-se ao mesmo conceito.

As regiões definidas pelo **VirtualLoop** são as seguintes:

- `CalibRegion`: região de calibração da câmera, utilizada para corrigir a perspectiva e calibrar as distâncias em metros para a medição de velocidade. Existe apenas uma por fluxo de vídeo
- `EntryRegion`: região de entrada, utilizada para demarcar o início do processo de rastreamento dos objetos. Deve existir ao menos 1 por fluxo para que ocorra o processamento
- `ActiveRegion`: região ativa, utilizada para demarcar a área final de uma trajeitória iniciada em uma `EntryRegion`
- `AbandonDetRegion`: região de detecção de objetos abandonados. Diferetemente da `ActiveRegion`, sua detecção não está atrelada a um deslocamento proveniente de uma `EntryRegion`



Demarcação da CalibRegion
Esquerda: *a região de calibração deve ser escolhida no plano do solo e deve ocupar a maior área útil possível*
Direita: *a região de calibração deve ser formada por um quadrilátero de lados ortogonais quando vistos de um ponto perpendicular ao plano do solo (visto de cima)*

2.1. Casos de Uso

A biblioteca **VirtualLoop** foi desenvolvida para ser utilizada nas mais diversas aplicações de detecção de movimento. Embora existam limitaçoes técnicas (ver **Limitações de Uso**), o **VirtualLoop** pode ser usado com câmeras panorâmicas, laterais ou fechadas. O **VirtualLoop** é robusto a situações de chuva pesada, neblina leve, sombra leve e operação noturna com iluminação pública.

Alguns casos de uso mais comuns para a biblioteca **VirtualLoop**, sem entretanto ficar restrita a eles, são:

- Detecção de parada sobre faixa de pedestres
- Detecção de avanço de sinal vermelho
- Detecção de conversão proibida
- Detecção de formação de fila, congestionamento e incidentes
- Detecção de objetos esquecidos ou abandonados
- Contagem de veículos
- Medição de ocupação de vias
- Medição de velocidade em aplicações não metrológicas
- Disparo para captura de imagem
- Detecção de movimento em geral

A figura a seguir apresenta 3 cenários diferentes de uso para o **VirtualLoop**. A área semitransparente branca delimita a região de inicio do rastreamento dos objetos enquanto os quadriláteros azuis e verdes marcam os laços virtuais.

Exemplos de aplicação do VirtualLoop
Vista lateral: *detecção de parada sobre faixa de pedestres, avanço de sinal*
Vista fechada: *contagem de veículos, disparo para captura de imagem*
Pórtico/Panorâmica: *contagem de veículos, ocupação, formação de fila, congestionamento e incidentes*

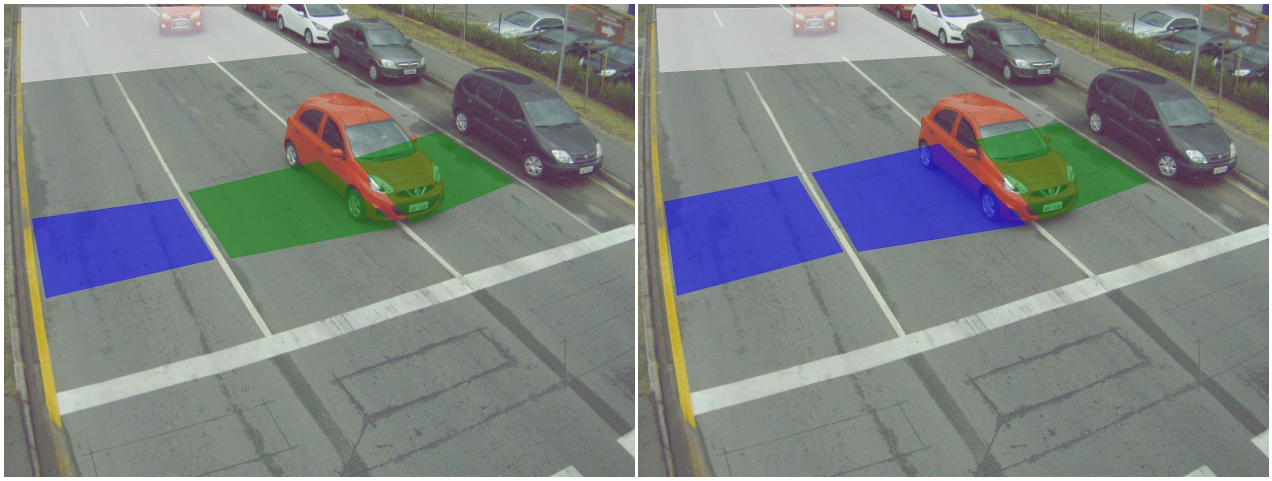
2.2. Princípio de Funcionamento

O **VirtualLoop** opera detectando e rastreando objetos que se movimentam ao longo de uma sequência de imagens. Para tanto é necessário definir uma região para o início desse processo rastreamento. Esta região, denominada região de entrada (`EntryRegion`), define e delimita o sentido de deslocamento dos objetos. Um objeto passa a ser seguido ao passar por esta região e é detectado quando intercepta um ou mais laços virtuais (`ActiveRegion`). A escolha da região de entrada garante, por exemplo, que em uma aplicação de detecção de parada de veículos sobre faixa de pedestres o movimento dos pedestres não seja confundido com o movimento dos veículos. Nesta situação a `EntryRegion` é posicionada na via e apenas os objetos que passam por ela têm suas trajetórias avaliadas quando interceptam o laço virtual sobre a faixa de pedestres.

O **VirtualLoop** pode operar em dois modos principais:

2.2.1 Modo Contagem

Este é o modo padrão de funcionamento do **VirtualLoop**, sendo recomendado para a maior parte dos casos de uso da biblioteca, este modo tem como premissa garantir que a contagem de veículos a passar por determinada via seja a mais fiel a uma contagem manual possível, assim não se limitando a características e limitações do laço magnético, como veículos entre faixas, motocicletas, etc.



Exemplo de diferença entre o Laço Magnético e o VirtualLoop
Esquerda: *laço magnético atua de maneira simultânea, dada sua sensibilidade e presença do veículo em ambas as faixas, há uma contagem extra*
Direita: *o VirtualLoop sendo capaz de atribuir o veículo a uma única faixa, assim contando corretamente apenas um veículo*

Uma `ActiveRegion` transiciona seu estado dependendo da quantidade e da intensidade do movimento detectados. Os estados possíveis e suas transições estão detalhados no diagrama da seção **enum JRegionState**. De maneira simplificada, toda `ActiveRegion` inicia no estado `IDLE`, indicando a ausência de movimento na região. Se algum objeto passar pela `EntryRegion` e se deslocar pela região delimitada pela `ActiveRegion`, esta irá transicionar para o estado `ACTIVE`, indicando que há movimento dentro da região. Se o objeto ao entrar na região a reduzir sua velocidade até que essa seja inferior a um deslocamento mínimo, a região irá para o estado de `HOLDING`. Este estado indica que existe a possibilidade do objeto estar parando na região ativa. Se este objeto permanecer com o deslocamento abaixo do mínimo por um tempo definido pelo usuário, o estado da região mudará para `STOPPED`, indicando que um objeto está parado na `ActiveRegion`. Caso o objeto se movimente novamente a região irá para o estado `ACTIVE`, retornando para `IDLE` quando o objeto sair completamente da região. Na transição da região para o estado de `IDLE` são computados os valores de contagem de veículos, tanto o total de contagem para a região, bem como o número de veículos que passaram durante a ativação.

> A histerese garantida pelo estado de `HOLDING` diminui falsas detecções de parada em situações de congestionamento. Nestas situações, é comum que os veículos acelerem e freiem bruscamente em um curto intervalo de tempo, o que geraria transições de `ACTIVE` para `STOPPED`. Como indicado em **enum JRegionState**, é possível desabilitar a histerese e fazer com que o laço passe diretamente de `ACTIVE` para `STOPPED`.

A figura a seguir mostra uma sequência típica de funcionamento do **VirtualLoop**. A `EntryRegion` foi colocada à esquerda da imagem para detectar veículos no sentido de aproximação com a câmera. Vemos na sequência que o veículo passa pela `EntryRegion` e se desloca até a `ActiveRegion`, incialmente em `IDLE`. Assim que o veículo entra na região do laço este passa para o estado `ACTIVE`. Já dentro do laço o veículo diminui a sua velocidade e para completamente, gerando os estados de `HOLDING` e `STOPPED`. Por fim, o veículo volta a se movimentar, transicionando o laço para `ACTIVE`, e sai da região retornando o laço para `IDLE`. A `ActiveRegion` também possui um parâmetro de tempo expiração de ativação em estado `STOPPED`, ou seja, em caso de falha da biblioteca, ou em caso de um veículo parado por muito tempo sobre a região de ativação, a biblioteca forçadamente retorna o estado da `ActiveRegion` para `IDLE`. Mais inforções sobre o parâmetro de expiração se encotram em **enum JVirtualLoopProperty**.

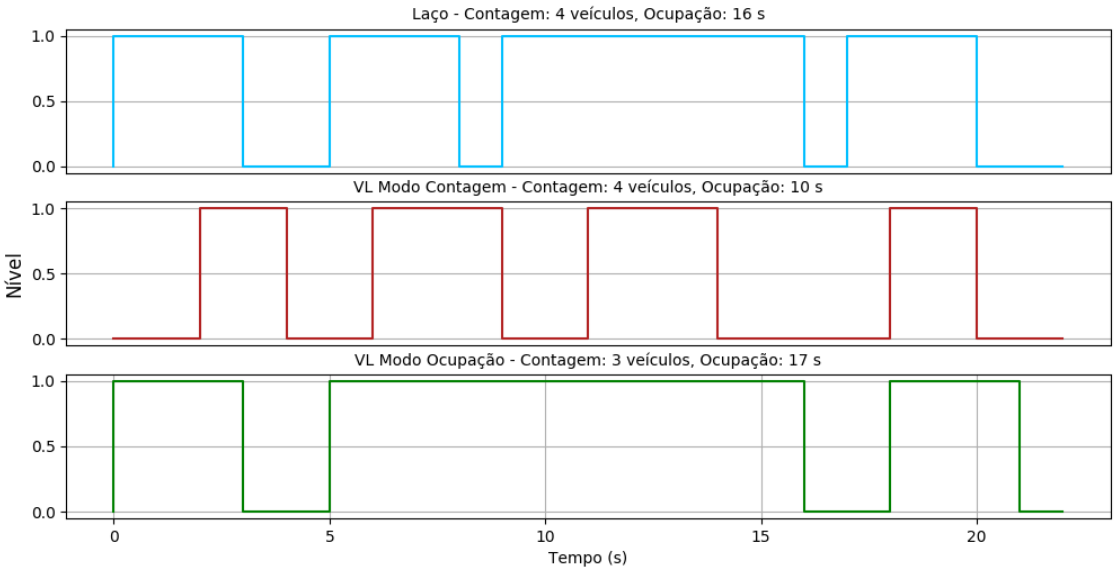
Funcionamento típico do VirtualLoop

*A **EntryRegion** é a região branca no canto esquerdo da imagem e a **ActiveRegion** é o quadrilátero no centro*
*As cores do quadrilátero indicam os estados da região: **IDLE** (azul), **ACTIVE** (verde), **HOLDING** (laranja) e **STOPPED** (vermelho)*

2.2.2 Modo Ocupação

Este modo alternativo de operação da biblioteca tem como objetivo garantir que o tempo de ativação do laço seja o mais próximo possível de um laço magnético equivalente na via. O modo de ocupação pode ser ativado através do arquivo de preferências `jl_tracker_preferences.json` (veja a seção **Arquivo de Preferências** com a opção `VL_DEFAULT_MODE` (0 - Contagem, 1 - Ocupação) e a borda da `ActiveRegion` a ser adotada pelo método com a opção `AR_CHOSEN_BOUNDARY` (0 - Borda Superior, 1 - Borda Direita, 2 - Borda Inferior, 3 - Borda Esquerda), vide **imagem**. Os valores de `PERSPECTIVE_CONSTANT` são utilizados no ajuste fino da largura do pulso de ativação da região.

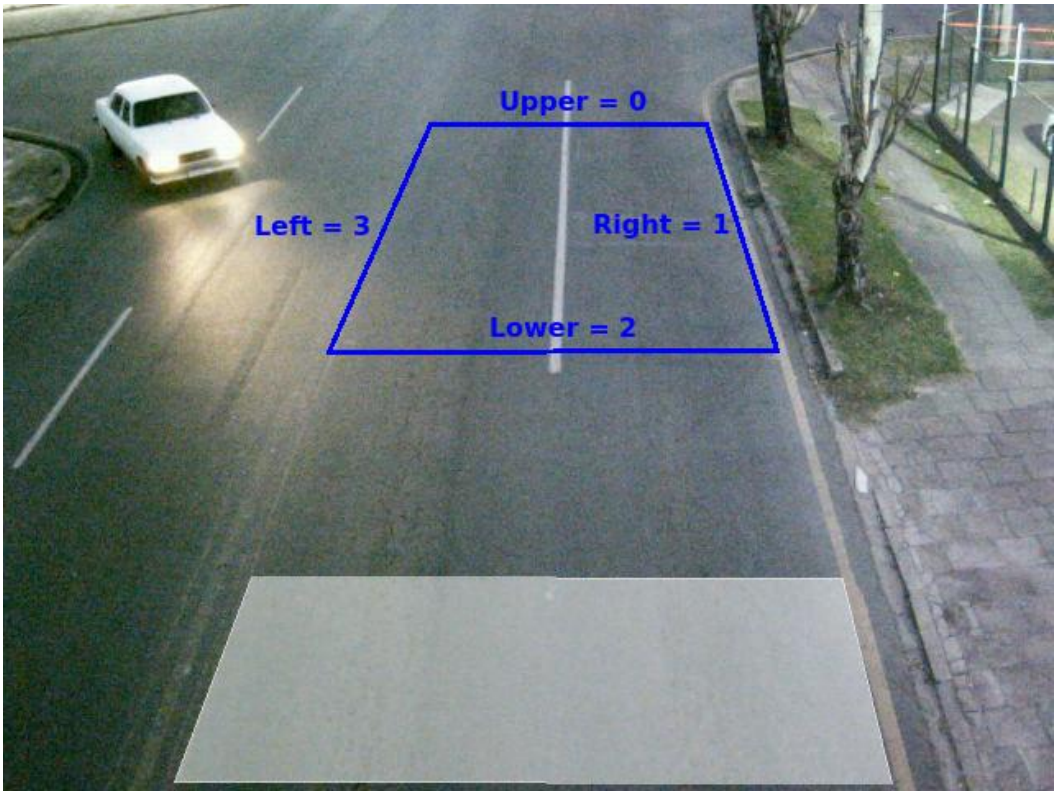
```
{
        "jidosha-light": {
            "object_detection": {
                "PERSPECTIVE_CONSTANT_ALPHA" : 0.55,
                "PERSPECTIVE_CONSTANT_BETA" : 0
            },
            "config": {
                "VL_DEFAULT_MODE" : 1,
                "AR_CHOSEN_BOUNDARY" : 0
            }
        }
    }
```



Comparativo entre modos de operação

Laço Sinal de referência, representando o comportamento de nível de um laço magnético físico instalado em uma determinada via
VL Contegem Em modo contagem, a biblioteca prioriza o número final de bordas de descida do sinal de ativação, não necessáriamente fidelizando os instantes ou a duração do pulso de ativação
VL Ocupação Em modo ocupação, a biblioteca prioriza replicar o tempo de ativação do laço indutivo, não necessariamente representando o número obeservado de pulsos

Uma diferença fundamental entre esse modo e o modo contagem se dá no uso da `ActiveRegion`. Quando em modo ocupação, a região ativa assumirá apenas os estados de `ACTIVE` e `IDLE`, onde sua ativação não se dá pela intensidade do movimento do veículo dentro da região, mas sim pela passagem do mesmo por uma da bordas, definida pelo usuário através do arquivo de preferências, no momento de saída da `ActiveRegion`. As bordas da região são definidas por sua proximidade das arestas da imagem, superior, direita, inferior e esquerda. A opção de ocupação também permite o uso de constantes de ajuste para correção do erro de perspectiva inerente às diferenças entre laços virtuais e magnéticos. Este modo assume um sentido único de via, ou seja, sendo selecionada uma borda pelo arquivo de preferências, todas as `ActiveRegion` desenhadas pelo usuário partilharam a mesma borda de ocupação.



ActiveRegion em modo ocupação

A ActiveRegion agora é tratada pelo conjunto de suas bordas. Neste cenário, onde a EntryRegion encontra-se na parte inferior da imagem, a borda correta a

ser adotada é a Upper

2.3. Limitações de Uso

Para que o **VirtualLoop** opere de forma satisfatória algumas condições devem ser atendidas. A mais importante delas está relacionada ao posicionamento da câmera, que deve permanecer estática em relação ao ambiente e deve possuir uma abertura de lente compatível com as dimensões dos objetos a serem detectados (objetos muito grandes ou muito pequenos devem ser evitados). Pequenas oscilações e trepidações causadas pelo vento ou pelo fluxo de veículos pesados são permitidas. Além disso, a câmera deve ser capaz de transmitir o fluxo de vídeo no formato MJPEG (Motion JPEG).

Outro ponto importante para garantir o bom funcionamento do **VirtualLoop** é a qualidade da imagem e a taxa de frames da câmera. A imagem deve estar com foco ajustado e o contraste entre os objetos e o ambiente deve ser razoável. Imagens muito desfocadas ("borradas") e clarões prejudicam o rastreamento e a detecção. Não é necessária uma resolução alta de imagem, sendo 640x480 pixels suficientes para a maioria das aplicações. A taxa de quadros mínima necessária varia com a velocidade da via, sendo recomendável ao menos 15 FPS para uso em vias com velocidade média de 60 Km/h.

Superfícies altamente refletivas (reflexão especular) como espelhos d'água ou metais polidos podem causar detecções indesejadas, uma vez que o processo de rastreamento pode não distinguir o objeto real do seu reflexo. **Sombras rígidas**, aquelas que causam a saturação da superfície incidente, também podem causar detecções inesperadas.

A lista abaixo resume os principais requisitos para o bom funcionamento do **VirtualLoop**.

- Posicionamento estático da câmera
- Fluxo de vídeo no formato MJPEG
- Taxa de frames compatível com a velocidade da via (15 FPS ou mais)
- Compatibilidade da abertura da lente com o tamanho dos objetos
- Boa iluminação do ambiente (funcionamento noturno apenas com iluminação pública ou se o veículo estiver com o farol aceso)
- Ausência de superfícies altamente refletivas
- Ausência de clarões, como o sol batendo no teto do veículo e estourando a imagem
- Ausência de sombras rígidas
- Ausência de oclusão (objetos rastreados que forem ocludidos serão perdidos)

O VirtualLoop foi concebido para trabalhar com câmeras estáticas e fixas. Seu uso não é recomendado em operações móveis.

Comparação entre uma sombra leve e uma sombra rígida
A sombra leve é semitransparente, permitindo que a textura do asfalto continue visível, já a sombra rígida provoca a saturação da região do asfalto, tornando dificil a separação entre o veículo e o seu entorno

Existem parâmetros do algoritmo de detecção do **VirtualLoop** que podem ser customizados para atender requisitos específicos de uma determinada instalação. Dentre estes parâmetros estão o aumento da robustez com sombras rígidas, diminuição da carga de processamento e melhoria da detecção em aplicações com baixa luminosidade. Para maiores informações entre em contato com o suporte através do email **contato@pumatronix.com.br**.

2.3.1. Uso em máquinas virtuais (VMs)

Embora seja possível a utilização de nossas bibliotecas em máquinas virtuais, havendo inclusive relatos de sucesso por parte de alguns clientes, seu uso é desencorajado e não homologado. A **Pumatronix Equipamentos Eletrônicos** não dá garantias de funcionamento e de suporte para o uso de seus produtos em máquinas virtuais.

3. Guia de Uso

Este capítulo traz as informações necessárias para integrar a biblioteca **VirtualLoop** a uma aplicação. É necessário ressaltar que o **VirtualLoop** opera em uma arquitetura cliente/servidor, interligados via rede TCP/IPv4 (veja a seção **Arquitetura de software** para maiores informações). Nos casos onde cliente e servidor ocupam a mesma máquina física, a comunicação ocorre via *loopback*.

3.1. Condições de Uso

O kit de desenvolvimento de software (SDK) do **VirtualLoop** é distribuído através de um conjunto de bibliotecas de software (.so e .dll) com *interface de programação de aplicação* (API) em linguagem C. As arquitetura de hardware suportadas são x86 32 e 64 bits (i686 e x86_64) com sistema operacional Linux ou Windows. O suporte a outras linguagens de programação é possível através da escrita de *wrappers* ou *binds* a partir da API C.

> NOTA: Por se comunicarem via rede TCP/IPv4, cliente e servidor **VirtualLoop** podem ser executados em ambientes heterogênios e distribuídos. O servidor pode estar em uma máquina com Windows Server enquanto o cliente em outra com Ubuntu, por exemplo.

Para o correto funcionamento da API servidor é necessário o uso do *hardkey* (chave de segurança) que acompanha a biblioteca. O *hardkey* deverá estar conectado à USB do equipamento onde o servidor será executado, caso contrário, o servidor não aceitará nenhuma conexão. A API cliente, por não executar nenhum tipo de processamento, não necessita de *hardkey*. Entretanto, o número máximo de clientes simultâneos que um único servidor **VirtualLoop** pode processar é limitado pela licença adquirida. Existem duas versões de *hardkey*, uma de demonstração e outra para uso geral, sendo que a versão de demonstração tem data de validade. Quando a data de validade desta expira, a bilbioteca automaticamente para de funcionar.

Para saber mais sobre o suporte em outras arquiteturas, como ARMv7, e outras linguagens como, Java, Python ou C#, ou ainda solicitar a compra de uma licença entre em contato com a **Pumatronix Equipamentos Eletrônicos** através do email **contato@pumatronix.com.br**.

3.2. Instalação

3.2.1. Pré requisitos do sistema

3.2.1.1. Software

- Sistema operacional
  - Linux 32/64 bits com gcc 4.9 ou superior e glibc 2.21 ou superior
  - Windows 32/64 bits versão 7 ou superior
- Firewall configurado para permitir conexões na porta TCP escolhida
- 7zip: **http://www.7-zip.org/**

3.2.1.2. Hardware

- Servidor
  - Porta USB (utilizada pelo *hardkey*)
  - Conexão Ethernet (caso o servidor aceite conexões externas)
  - CPU com 4 cores ou mais
  - 4GB ou mais de memória RAM
- Cliente
  - 1GB ou mais de memória RAM

3.2.2. Descompactando o SDK

O SDK do **VirtualLoop** é distribuído através de um arquivo compactado no formato 7zip. A descompactação deste arquivo requer uma senha, fornecida junto com o email contendo as instruções de download deste SDK. Caso você tenha problemas com a descompactação do SDK contate o suporte pelo email **contato@pumatronix.com.br**.

Para descompactar o SDK em ambiente Linux, utilize o seguinte comando a partir de um terminal (substitua os campos e <Virtual_Loop_PC_LINUX_64_vX.Y.Z.7z> pela senha fornecida por email e pelo nome correto do pacote).

```
7z x -p<SENHA_FORNECIDA> <Virtual_Loop_PC_LINUX_64_vX.Y.Z.7z>
```

3.2.3. Permissões do *hardkey*

> Esta configuração só é necessária para a versão Linux do SDK. Caso você esteja em ambiente Windows, vá para para **Configuração das variáveis de ambiente**.

Para o correto funcionamento do *hardkey* USB no Linux, as permissões de acesso do **udev** devem ser alteradas. Adicione a seguinte linha:

```
ATTRS{idVendor}=="0403", ATTRS{idProduct}=="c580", MODE="0666"
```

ao final do arquivo correspondente a sua distribuição Linux:

```
Centos 5.2/5.4:          /etc/udev/rules.d/50-udev.rules
Centos 6.0 em diante:    /lib/udev/rules.d/50-udev-default.rules
Ubuntu 7.10:             /etc/udev/rules.d/40-permissions.rules
Ubuntu 8.04/8.10:        /etc/udev/rules.d/40-basic-permissions.rules
Ubuntu 9.04 em diante:   /lib/udev/rules.d/50-udev-default.rules
openSUSE 11.2 em diante: /lib/udev/rules.d/50-udev-default.rules
```

Caso a distribuição seja Debian, adicione as linhas:

```
SUBSYSTEM=="usb_device", MODE="0666"
SUBSYSTEM=="usb", ENV{DEVTYPE}=="usb_device", MODE="0666"
```

ao final do arquivo:

```
Debian 6.0 em diante:    /lib/udev/rules.d/91-permissions.rules
```

Para instruções de como habilitar o *hardkey* em outras distribuições Linux, entre em contato com o **contato@pumatronix.com.br**.

3.2.4. Variáveis de ambiente

3.2.4.1 Configuração do LD_LIBRARY_PATH

> Esta configuração só é necessária para a versão Linux do SDK. Para a versão Windows, faça uma cópia da DLL para a pasta onde está o executável ou para a pasta system32

Para que a aplicação seja capaz de encontrar as bibliotecas do **VirtualLoop** deve-se adicionar ao PATH o caminho do diretório lib do SDK. Em ambiente Linux isto pode ser feito através da variável de ambiente `LD_LIBRARY_PATH`, como exemplificado abaixo (substitua pelo caminho absoluto da instalação do SDK).

```
export LD_LIBRARY_PATH=<SDK-ROOT-DIR>/lib
```

3.2.4.2 Configuração do LOG

A biblioteca **VirtualLoop** possui um sistema de log que vem habilitado por padrão. Para inibir as mensagens é necessário definir a variável de ambiente `JL_LOGCFG` como o valor `DISABLED`. Em ambiente Linux basta abrir um terminal e executar o seguinte comando:

```
export JL_LOGCFG=DISABLED
```

Quando o log está habilitado, por padrão, as mensagens são enviados pelo `stdout` do sistema. Entretanto, é possível customizar este comportamento através de um arquivo de configuração, cujo formato é brevemente descrito na sequência. Para que as configurações definidas no arquivo sejam utilizadas pela biblioteca, basta definir a variável de ambiente `JL_LOGCFG` com o caminho absoluto do arquivo de configuração a ser utilizado. Para que as configurações tenham efeito, deve-se definir a variável antes da execução da aplicação. Para maiores informações sobre como modificar o comportamento do sistema de log do **VirtualLoop** contate **contato@pumatronix.com.br**.

```
#
# VirtualLoop JLog configuration file sample
#
# This is a comment line in a JLog configuration file, it **must** start with one #
#
# Entry format (one per line):
#   TOPIC; LEVEL; TAG_FMT, FILES (comma separated); SIZES  (comma separated)
#
# Especial Files
#   [STDOUT] - prints to the screen, SIZE must be 0
#
LICENSE   ; INFO          ; SIMPLE_TS    ; [STDOUT], lic.info.log       ; 0, 2MB
LICENSE   ; WARN          ; SIMPLE_TS    ; [STDOUT], lic.warn.log       ; 0, 2MB
LICENSE   ; NOTICE        ; SIMPLE_TS    ; [STDOUT], lic.notice.log     ; 0, 2MB
LICENSE   ; CRITICAL      ; SIMPLE_TS    ; [STDOUT], lic.critical.log   ; 0, 2MB
HARDWARE  ; INFO          ; SIMPLE_TS    ; [STDOUT]                     ; 0
HARDWARE  ; CRITICAL      ; SIMPLE_TS    ; [STDOUT]                     ; 0
JLIB      ; INFO          ; SIMPLE_TS    ; jlib.info.log                ; 1MB
JLIB      ; CRITICAL      ; SIMPLE_TS    ; jlib.critical.log            ; 512KB
```

3.3. Estrutura do SDK

O **VirtualLoop** não foi projetado como uma aplicação *standalone* e sim como uma biblioteca a ser integrada em outras aplicações. Dessa forma, o SDK do **VirtualLoop** é distribuido como um conjunto de shared libraries (.so e .dll) e seus headers em linguagem C (um bind Delphi é fornecido com a versão Windows 32 bits do SDK).

A API é dividida em três submodulos: **API Cliente**, **API Servidor** e **API Utilitária**. As duas primeiras implementam, respectivamente, um cliente e um servidor do **VirtualLoop**, enquanto a última provê funcionalidades básicas de leitura do *hardkey* e recepção de vídeo no formato MJPEG. Para um maior detalhamento das APIs, veja o capítulo **APIs de usuário** deste manual.

O SDK acompanha ainda duas aplicações de exemplo (código fonte e binários pré-compilados) que podem ser utilizadas como base na implementação de um par cliente/servidor **VirtualLoop**. A seção **Exemplos Básicos** deste manual traz também um passo-a-passo de como implementar um cliente e um servidor **VirtualLoop**.

3.3.1. Árvore de arquivos versão Linux

```
├── include
│   └── ptx
│       ├── common
│       │   ├── ptx_codes.h
│       │   └── ptx_defines.h
│       ├── ptx_common.h
│       ├── virtualLoop
│       │   ├── vl_api_client.h
│       │   ├── vl_api_common.h
│       │   ├── vl_api_local.h
│       │   └── vl_api_server.h
│       └── virtualLoop.h
├── lib
│   ├── libptx_virtualLoopJava.so
│   └── libptx_virtualLoop.so
├── res
│   └── manual_virtual_loop.pdf
├── sample
│   ├── bin
│   │   ├── VirtualLoopClient
│   │   └── VirtualLoopServer
│   └── src
│       ├── VirtualLoopClient.c
│       └── VirtualLoopServer.c
└── wrapper
    └── java
        └── br
            └── com
                └── pumatronix
                    ├── common
                    │   └── PtxCommon.java
                    └── vl
                        ├── VLClient.java
                        ├── VLCommon.java
                        ├── VLSample.java
                        └── VLServer.java
```

3.3.2. Árvore de arquivos versão Windows

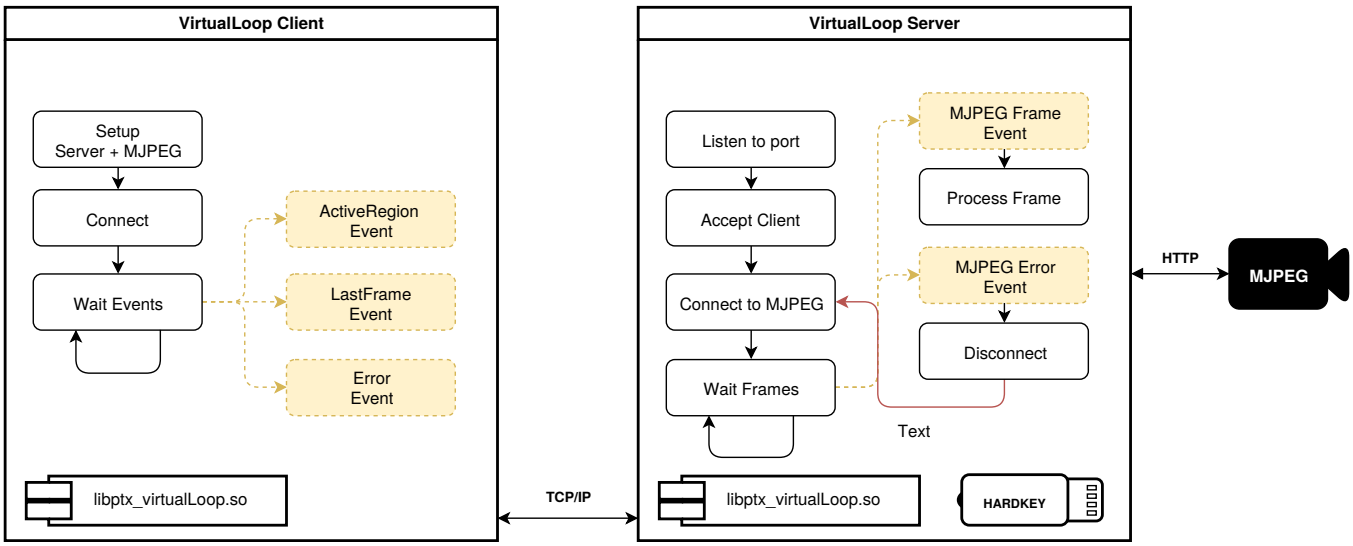
```
├── include
│   └── ptx
│       ├── common
│       │   ├── ptx_codes.h
│       │   └── ptx_defines.h
│       ├── ptx_common.h
│       ├── virtualLoop
│       │   ├── vl_api_client.h
│       │   ├── vl_api_common.h
│       │   ├── vl_api_local.h
│       │   └── vl_api_server.h
│       └── virtualLoop.h
├── lib
│   ├── ptx_virtualLoop.dll
│   ├── ptx_virtualLoopJava.dll
│   ├── ptx_virtualLoopJava.lib
│   └── ptx_virtualLoop.lib
├── res
│   └── manual_virtual_loop.pdf
├── sample
│   ├── bin
│   │   ├── VirtualLoopClient.exe
│   │   └── VirtualLoopServer.exe
│   └── src
│       ├── VirtualLoopClient.c
│       └── VirtualLoopServer.c
└── wrapper
    ├── delphi
    │   ├── ptx
    │   │   ├── common
    │   │   │   └── ptx_common.pas
    │   │   └── virtualLoop
    │   │       ├── vl_client.pas
    │   │       └── vl_server.pas
    │   └── VirtualLoopClient.dpr
    └── java
        └── br
            └── com
                └── pumatronix
                    ├── common
                    │   └── PtxCommon.java
                    └── vl
```

```
├── VLClient.java
├── VLCommon.java
├── VLSample.java
└── VLServer.java
```

3.4. Arquitetura de software

O **VirtualLoop** possui uma arquitetura assíncrona baseada em eventos, operando através de uma topologia cliente/servidor. Todo o processo de análise de vídeo ocorre na instância do servidor enquanto o cliente aguarda os eventos. A API servidor possui um nível de abstração bastante elevado, necessitando poucas linhas de código para sua integração. Internamente, o servidor paraleliza o processamento de múltiplos clientes e monitora a conectividade com a câmera, mantendo uma fila de imagens e reconectando quando necessário.

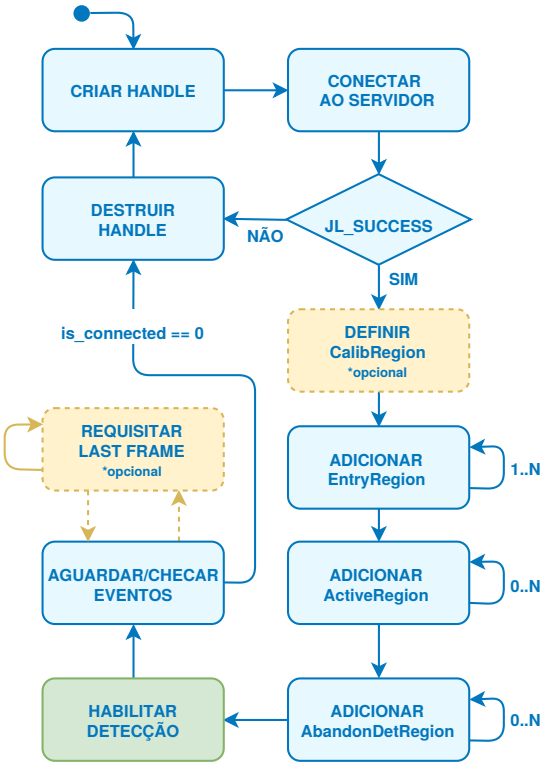
Um cliente **VirtualLoop** necessita especificar as configurações do servidor, registrar funções (*callbacks*) que serão chamadas quando ocorrem os eventos, conectar ao servidor e definir as regiões de interesse. Depois de conectado, o cliente pode ainda solicitar imagens da câmera para o servidor e checar o estado da conexão.



Arquitetura Cliente/Servidor VirtualLoop

O servidor VirtualLoop é responsável por gerenciar o fluxo MJPEG e processar os quadros de vídeo. Caso ocorra algum erro que leve o fluxo MJPEG a desconectar, o servidor avisa o cliente e automaticamente tenta reestabelecer a conexão.

O fluxograma detalhado da configuração de um cliente VirtualLoop é apresentado a seguir. É importante notar que a adição das regiões deve respeitar a seguinte ordem: `CalibRegion`, `EntryRegion`, `ActiveRegion` e por fim `AbandonDetRegion`. Assim, ao menos uma `EntryRegion` deve existir para que se possa adicionar uma `ActiveRegion` ou uma `AbandonDetRegion`. A definição da `CalibRegion` só é obrigatória para aplicações com medição de velocidade.



Fluxograma de configuração de um cliente VirtualLoop

3.5. Exemplos Básicos

Para um conjunto de exemplos mais completos, veja a pasta `sample/src` do SDK.

3.5.1. Servidor

Para criar uma aplicação que inicia um servidor **VirtualLoop** na porta 60000 e aceita até 2 clientes simultâneos, crie um arquivo chamado `server.c` na raiz do SDK e insira o seguinte código:

```
#include <stdlib.h>
#include <stdio.h>
#include "ptx/virtualLoop.h"
int main(void) {
    const int port = 60000;
    const int nclients = 2;
    JVirtualLoopServer* server = jl_vl_create_server(port, nclients);
    if(server != NULL) {
        while( getchar() != 'q');
        jl_vl_destroy_server(server);
    }
    return 0;
}
```

Para compilar com o gcc e executar, a partir de um terminal no Linux, execute a seguinte sequencia de comandos:

```
gcc server.c -I include/ -L lib/ -l ptx_virtualLoop -o server
LD_LIBRARY_PATH=lib/ ./server
```

Assim que o servidor for iniciado, deve-se ler na saída do terminal:

```
[2017:07:06 18:05:55.750234 : LOGGER   : 0x0001 : INFO] -> JLib log session started
[2017:07:06 18:05:55.750318 : JLIB     : 0x0004 : INFO] -> JLib singleton created
[2017:07:06 18:05:55.794436 : JLIB     : 0x0001 : INFO] -> JLib network module started
[2017:07:06 18:05:55.794511 : MSGSERVER : 0x0001 : INFO] -> Started server at port 60000
```

Para encerrar o servidor basta apertar a sequência `q<ENTER>`.

3.5.2. Cliente

Para iniciar um cliente **VirtualLoop** é necessario definir os parâmetros de configuração do servidor e do fluxo MJPEG, registrar as callbacks dos eventos, conectar ao servidor, definir as regiões de interesse e habilitar a detecção.

Para compilar o exemplo fornecido com gcc e executá-lo a partir de um terminal no Linux, execute a seguinte sequência de comandos:

```
gcc sample/src/VirtualLoopClient.c -I include/ -L lib/ -l ptx_virtualLoop -o client
LD_LIBRARY_PATH=lib/ ./client
```

3.6 Arquivo de Preferências

Durante a inicialização do servidor do `VirtualLoop`, a biblioteca consulta seu diretório local em busca de um arquivo chamado `jl_tracker_preferences.json`. Este arquivo é também recarregado periódicamente pela aplicação a cada 30 segundos e tem como função principal alterar as constantes/parâmetros globais fundamentais da biblioteca. Abaixo temos um exemplo com parâmetros globais acessíveis via arquivo de preferências. Nota-se que alguns destes valores também podem ser alterados via chamada de API, valores assim alterados terão precedência sobre valores presentes no arquivo de preferências.

```
{
        "jidosha-light" : {
            "config" : {
                "VL_DEFAULT_MODE" : 0,
                "AR_CHOSEN_BOUNDARY" : 0,
                "AR_TIMEOUT" : 300,
                "ABANDON_MIN_NUM_FRAMES" : 150
            },
            "object_detection" : {
                "TRACK_ENABLE_VEHICLE_TYPE_DETECTION" : true,
                "AUTHORIZED_VEHICLE_TYPES" : 30,
                "TRACKING" : true,
                "NUM_THREADS" : 2,
                "PERSPECTIVE_CONSTANT_ALPHA" : 0.55,
                "PERSPECTIVE_CONSTANT_BETA" : 0
            }
        }
    }
```

A tabela abaixo descreve cada um dos campos do arquivo de preferências em maior detalhe.

- Campo `config`:

Campo	Atualização	Descrição
VL_DEFAULT_MODE	Subida do servidor	Define o **modo de operação** do `VirtualLoop`
AR_CHOSEN_BOUNDARY	Criação da `Active Region`	Define a borda a ser adotada pelo software em **modo ocupação**
AR_TIMEOUT	Criação da `Active Region`	Tempo de expiração forçada da ativação em segundos
ABANDON_MIN_NUM_FRAMES	Criação da `AbandonDet Region`	Número mínimo de frames até um objeto ser considerado abandonado por uma `AbandonDet Region`

- Campo `object_detection`:

Campo	Atualização	Descrição
TRACK_ENABLE_VEHICLE_TYPE_DETECTION	Subida do servidor	Habilita a função de detecção de tipo de veículo
AUTHORIZED_VEHICLE_TYPES	A cada 30s	Define quais tipos de veículos são capazes de gerar um evento de ativação
TRACKING	Subida do servidor	Permite o rastreamento de objetos (aumenta a precisão de contagem e detecção)
NUM_THREADS	Subida do servidor	Número máximo de threads alocadas para a detecção de veículos
PERSPECTIVE_CONSTANT_ALPHA	A cada 30s	Parcela proporcional do fator de perspectiva usado no ajuste fino do **modo ocupação**
PERSPECTIVE_CONSTANT_BETA	A cada 30s	Parcela constante do fator de perspectiva usado no ajuste fino do **modo ocupação**

4. APIs de usuário

Esta seção detalha as APIs do **VirtualLoop** presentes no SDK. Por padrão, as linguagens suportadas pelo SDK são C/C++ e Delphi.

Em caso de dúvidas sobre a integração ou suporte a outras linguagens, envie um email para **contato@pumatronix.com.br**.

4.1. API VirtualLoop C/C++

A API (Application Programming Interface) nativa da biblioteca está escrita em linguagem C, o que facilita a escrita de binds para outras linguagens. Toda a API C está exposta através de um conjunto de *headers* dentro da pasta **include** do SDK. A *calling convention* adotada para as chamadas da API e que também deve ser adotada pelas callbacks de usuário é a **stdcall** (Windows) ou **cdecl**(Linux), conforme definido pela macro **PTXAPI** em **ptx/common/defines.h**.

4.1.1. API Cliente

```
//============================================================================
    //  VIRTUAL LOOP ACTIVE REGION STATE
    //============================================================================
    typedef enum JRegionState {
        PTX_VL_REGION_STATE_IDLE                    = 0,
        PTX_VL_REGION_STATE_ACTIVE                  = 1,
        PTX_VL_REGION_STATE_HOLDING                 = 2,
        PTX_VL_REGION_STATE_LINE_STOP               = 3,
        PTX_VL_REGION_INVALID                       = 4,
        PTX_VL_REGION_ENUM_MAX
    } JRegionState;

    typedef enum JEventExtraField {
        PTX_VL_EVENT_SPEED                          = 0,
        PTX_VL_EVENT_COUNTER_TOTAL                  = 1,
        PTX_VL_EVENT_COUNTER_SAME_ORIGIN            = 2,
        PTX_VL_EVENT_VEHICLE_TYPE                   = 3,
        PTX_VL_EVENT_COUNTER_DELTA                  = 4,
        PTX_VL_EVENT_ENUM_MAX
    } JEventExtraField;

    typedef enum JVehicleTypes {
        PTX_VL_VEHICLE_TYPE_UNKNOWN                 = 0,
        PTX_VL_VEHICLE_TYPE_CAR                     = 1,
        PTX_VL_VEHICLE_TYPE_MOTORCYCLE              = 2,
        PTX_VL_VEHICLE_TYPE_TRUCK                   = 3,
        PTX_VL_VEHICLE_TYPE_BUS                     = 4,
        PTX_VL_VEHICLE_TYPE_ENUM_MAX
    } JVehicleTypes;

    typedef enum JVirtualLoopProperty {
        PTX_VL_PROPERTY_VEHICLE_TYPE_ENABLED        = 0,
        PTX_VL_PROPERTY_AUTHORIZED_VEHICLE_TYPES    = 1,
        PTX_VL_PROPERTY_ACTIVE_REGION_TIMEOUT       = 2,
        PTX_VL_PROPERTY_ENUM_MAX
    } JVirtualLoopProperty;

    //============================================================================
    // HELPER TYPES
    //============================================================================
    typedef struct Point2i {
        int32_t x;
        int32_t y;
    } Point2i;

    typedef struct JTimestamp {
        uint16_t year;
        uint16_t month;
        uint16_t day;
        uint16_t hour;
        uint16_t min;
        uint16_t sec;
        uint16_t msec;
        uint16_t pad;   /* unused field */
    } JTimestamp;

    //============================================================================
    // MAIN TYPES
    //============================================================================
    typedef struct JVirtualLoopHandle JVirtualLoopHandle;

    typedef struct JVirtualLoopHandleStatus {
        int is_connected;
    } JVirtualLoopHandleStatus;

    typedef struct JVirtualLoopEntryRegion {
        uint32_t id;
        Point2i points[4];
    } JVirtualLoopEntryRegion;

    typedef struct JVirtualLoopActiveRegion {
        uint32_t id;
        Point2i points[4];
        uint32_t deb_frames;
        uint32_t stop_frames;
        uint32_t sense_thresh;
    } JVirtualLoopActiveRegion;

    typedef struct JVirtualLoopAbandonDetRegion {
        uint32_t id;
        Point2i points[4];
```

```
} JVirtualLoopAbandonDetRegion;

typedef struct JVirtualLoopCalibRegion {
    Point2i points[4];  /* corners of a rectangle placed on the road plane in pixels */
    float width;        /* width in meters of the rectangle */
    float height;       /* height in meters of the rectangle */
} JVirtualLoopCalibRegion;

typedef struct JVirtualLoopAbandonObject {
    uint32_t id;
    uint32_t x;
    uint32_t y;
    uint32_t width;
    uint32_t height;
} JVirtualLoopAbandonObject;

typedef struct JVirtualLoopEvent {
    JVirtualLoopEntryRegion origin;
    JVirtualLoopActiveRegion target;
    JTimestamp timestamp;
    JRegionState state;
    void* extra;
} JVirtualLoopEvent;

typedef struct JVirtualLoopLicenseInfo
{
    uint64_t serial;
    char customer[64];
    int  maxThreads;
    int  maxConnections;
    int  state;
    int  ttl;
} JVirtualLoopLicenseInfo;

typedef struct JVirtualLoopServerInfo
{
    JVirtualLoopLicenseInfo license;
    int major;
    int minor;
    int release;
    char sha1[64];
} JVirtualLoopServerInfo;

typedef struct JVirtualLoopConfig {

    const char* mjpeg_url;
    const char* server_ip;
    int32_t server_port;
    int32_t frame_queue_cap;
    void* user_ctx;

    /* event callbacks */
    void (PTXAPI *on_event_cb)(
        const JVirtualLoopEvent* event,
        const uint8_t* jpeg_ptr,
        uint32_t jpeg_size,
        void* user_ctx
    );

    void (PTXAPI *on_abandon_cb)(
        const JVirtualLoopAbandonObject* regions,
        uint32_t regions_size,
        const uint8_t* jpeg_ptr,
        uint32_t jpeg_size,
        void* user_ctx
    );

    void (PTXAPI *on_error_cb)(
        int32_t error_id,
        void* user_ctx
    );

    void (PTXAPI *on_frame_cb)(
        const uint8_t* jpeg_ptr,
        uint32_t jpeg_size,
        void* user_ctx
    );

} JVirtualLoopConfig;

//================================================================================
// METHODS
//================================================================================
/* handle */
/* create/destroy */
PTXEXPORT JVirtualLoopHandle* PTXAPI jl_vl_create_handle();
PTXEXPORT void PTXAPI jl_vl_destroy_handle(JVirtualLoopHandle* handle);
#define jl_vl_clean_handle(handle) { if(handle) { jl_vl_destroy_handle(handle); handle = NULL; } }

/* requests */
PTXEXPORT int PTXAPI jl_vl_connect(JVirtualLoopHandle* handle, JVirtualLoopConfig* config);
PTXEXPORT int PTXAPI jl_vl_connect_info(JVirtualLoopHandle* handle, JVirtualLoopConfig* config, JVirtualLoopServerInfo* info
PTXEXPORT int PTXAPI jl_vl_disconnect(JVirtualLoopHandle* handle);
PTXEXPORT int PTXAPI jl_vl_disconnect_cb(JVirtualLoopHandle* handle);
PTXEXPORT int PTXAPI jl_vl_request_last_frame(JVirtualLoopHandle* handle);
PTXEXPORT int PTXAPI jl_vl_set_calib_region(JVirtualLoopHandle* handle, JVirtualLoopCalibRegion* cr);
PTXEXPORT int PTXAPI jl_vl_add_entry_region(JVirtualLoopHandle* handle, JVirtualLoopEntryRegion* er);
PTXEXPORT int PTXAPI jl_vl_add_active_region(JVirtualLoopHandle* handle, JVirtualLoopActiveRegion* ar);
PTXEXPORT int PTXAPI jl_vl_add_abandon_region(JVirtualLoopHandle* handle, JVirtualLoopAbandonDetRegion* abr);
PTXEXPORT int PTXAPI jl_vl_set_float_property(JVirtualLoopHandle* handle, JVirtualLoopProperty prop, float value);
PTXEXPORT int PTXAPI jl_vl_get_float_property(JVirtualLoopHandle* handle, JVirtualLoopProperty prop, float* value);
PTXEXPORT int PTXAPI jl_vl_enable_detection(JVirtualLoopHandle* handle);

/* status */
PTXEXPORT int PTXAPI jl_vl_get_handle_status(JVirtualLoopHandle* handle, JVirtualLoopHandleStatus* status);

/* events */
```

```
    PTXEXPORT int PTXAPI jl_vl_has_event_extra(const JVirtualLoopEvent* event, JEventExtraField field);
    PTXEXPORT float PTXAPI jl_vl_get_event_extra_float(const JVirtualLoopEvent* event, JEventExtraField field, float dvalue);

    /* utils */
    /* build info */
    PTXEXPORT int PTXAPI jl_vl_get_version(int* major, int* minor, int* release);
    PTXEXPORT const char* PTXAPI jl_vl_get_build_SHA1();
    /* session file */
    PTXEXPORT int PTXAPI jl_vl_load_session_file(JVirtualLoopHandle* handle, const char* filename);
```

4.1.1.1. Tipos

enum JRegionState

Descrição

Definição dos possíveis estados de uma **ActiveRegion**.

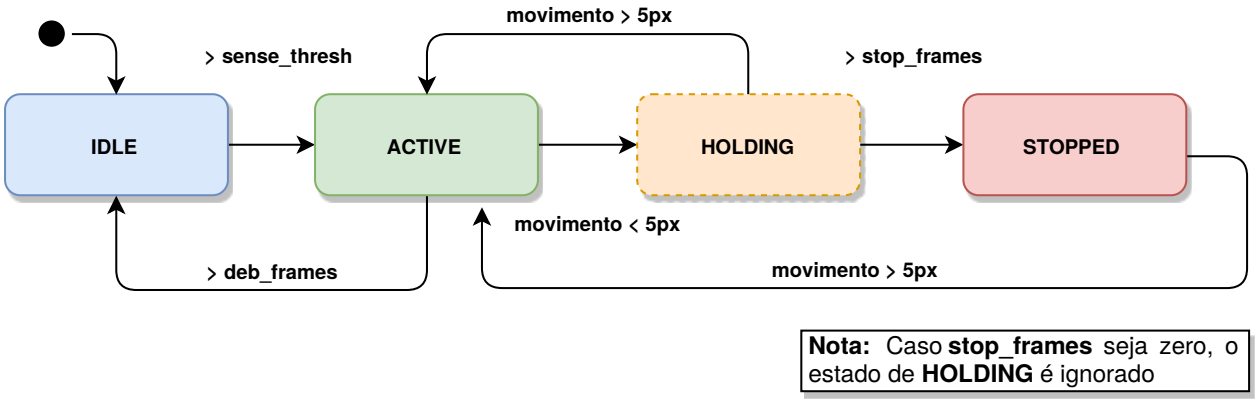


Diagrama de estados de uma ActiveRegion
*Para entender o significado dos parâmetros de transição ver **jl_vl_add_active_region***

Membros

- `PTX_VL_REGION_STATE_IDLE` : nenhum movimento detectado na região do laço
- `PTX_VL_REGION_STATE_ACTIVE` : movimento detectado na região do laço
- `PTX_VL_REGION_STATE_HOLDING` : parada detectada mas **stop_frames** ainda não excedido
- `PTX_VL_REGION_STATE_LINE_STOP` : parada detectada
- `PTX_VL_REGION_INVALID` : valor reservado para implementações futuras

enum JEventExtraField

Descrição

Definição dos possíveis campos adicionais da **struct JVirtualLoopEvent**.

Membros

- `PTX_VL_EVENT_SPEED` : velocidade média no momento do evento. Atualizada apenas nos eventos de `ACTIVE`
- `PTX_VL_EVENT_COUNTER_TOTAL` : valor acumulado de contagem. Atualizado nos eventos de `IDLE`
- `PTX_VL_EVENT_COUNTER_SAME_ORIGIN` : valor acumulado de contagem para eventos com a mesma origem do evento atual. Atualizado nos eventos de `IDLE`
- `PTX_VL_EVENT_VEHICLE_TYPE` : tipo do veículo que atravessa a região ativa - ver **enum JVehicleTypes**. Atualizada apenas nos eventos de `ACTIVE`
- `PTX_VL_EVENT_COUNTER_DELTA` : número de veículos contados durante o último ciclo de ativação. Atualizado nos eventos de `IDLE`

enum JVehicleTypes

Descrição

Definição das possíveis saídas de classificação dos veículos que passam por uma região ativa.

Membros

- `PTX_VL_VEHICLE_TYPE_UNKNOWN` : classe que define que o veículo não foi reconhecido
- `PTX_VL_VEHICLE_TYPE_CAR` : o veículo que atravessa a região ativa é do tipo carro
- `PTX_VL_VEHICLE_TYPE_MOTORCYCLE` : o veículo que atravessa a região ativa é do tipo moto

- `PTX_VL_VEHICLE_TYPE_TRUCK` : o veículo que atravessa a região ativa é do tipo caminhão
- `PTX_VL_VEHICLE_TYPE_BUS` : o veículo que atravessa a região ativa é do tipo ônibus

enum JVirtualLoopProperty

Descrição

Definição das possíveis configurações que podem ser realizadas no VirtualLoop.

Membros

- `PTX_VL_PROPERTY_VEHICLE_TYPE_ENABLED` : ativa ou desativa a classificação de tipo veicular (default: desativada)
- `PTX_VL_PROPERTY_AUTHORIZED_VEHICLE_TYPES` : Propriedade a ser utilizada como máscara para se ativar ou desativar determinada categoria de veículo. Uma categoria desativada não ativa o laço virtual e portanto não gera eventos. Os valores que habilitam cada categoria são da forma "0x1 << **JVehicleTypes**" e podem ser somados (default: 30 - todas as categorias habilitadas).

```
Exemplo:
    16 (PTX_VL_VEHICLE_TYPE_BUS)
    +8 (PTX_VL_VEHICLE_TYPE_TRUCK)
    +4 (PTX_VL_VEHICLE_TYPE_MOTORCYCLE)
    +2 (PTX_VL_VEHICLE_TYPE_CAR)
    = 30 (todas as categorias habilitadas)
    Nota: O valor da propriedade é um float e portanto não se deve utilizar operações bitwise.
```

- `PTX_VL_PROPERTY_ACTIVE_REGION_TIMEOUT` : número inteiro a ser fornecido em segundos que estabelece o tempo máximo de retenção do nível alto `STOPPED` da região ativa. Esse valor é estabelecido no momento de adição da região ativa ao cenário e não será atualizado em tempo real. Esse valor é global para todas as regiões ativas (default: 300s).

---

struct Point2i

Descrição

Struct C usada para representar um ponto 2D com coordenadas inteiras.

Membros

- `int32_t x` : coordenada x do ponto
- `int32_t y` : coordenada y do ponto

---

struct JTimestamp

Descrição

Struct C usada para retornar um timestamp. O timestamp retornado pela callback de evento é relativo ao servidor e não tem garantia de ser monotônico.

Membros

- `uint16_t year` : valor inteiro do ano [ex: 2017]
- `uint16_t month` : valor inteiro do mês do ano [1-12]
- `uint16_t day` : valor inteiro do dia do mês [1-31]
- `uint16_t hour` : valor inteiro da hora do dia [0-23]
- `uint16_t min` : valor inteiro do minuto [0-59]
- `uint16_t sec` : valor inteiro do segundo [0-59]
- `uint16_t msec` : valor inteiro do milisegundo [0-999]
- `uint16_t pad` : preenchimento adicionado para garantir o alinhamento da struct em 32 bits [sempre 0]

---

struct JVirtualLoopHandle

Descrição

Tipo opaco utilizado para armazenar o handle cliente do VirtualLoop.

Membros

Nenhum

---

struct JVirtualLoopHandleStatus

Descrição

Struct utilizada para retornar o estado de um cliente do VirtualLoop.

Membros

- `int is_connected` : indica o estado da conexão do handle com o servidor [0 disconectado, 1 conectado]

`struct JVirtualLoopEntryRegion`

Descrição

Struct utilizada para definir uma **EntryRegion**.

Membros

- `uint32_t id` : ID utilizado para identificar a região
- `Point2i points[4]` : conjunto de 4 pontos, em qualquer ordem, utilizados para delimitar o quadrilátero da região de entrada

---

`struct JVirtualLoopActiveRegion`

Descrição

Struct utilizada para definir uma **ActiveRegion**.

Membros

- `uint32_t id` : ID utilizado para identificar a região
- `Point2i points[4]` : conjunto de 4 pontos, em qualquer ordem, utilizados para delimitar o quadrilátero da região de entrada
- `uint32_t deb_frames` : número de frames utilizados para o debounce da região ativa. Valores típicos para este campo vão de **4** a **8**. Este parâmetro tem o intuito de reduzir multiplas detecções de veículos muito longos, como caminhões e ônibus
- `uint32_t stop_frames` : número de frames para que um evento de holding seja considerado uma parada. Este valor deve ser escolhido baseado na taxa de frames do fluxo Mjpeg
- `uint32_t sense_thresh` : valor de sensibilidade do laço no intervalo [1-30], sendo 1 mais sensível e 30 menos sensível. Este valor está relacionado a ocupação mínima do laço para que o movimento sejá considerado. Valores maiores devem ser utilizados para laços maiores. Aplicações noturnas ou com pouca iluminação devem utilizar valores menores

---

`struct JVirtualLoopAbandonDetRegion`

Descrição

Struct utilizada para delimitar uma região de detecção de objectos abandonados.

Membros

- `uint32_t id` : ID utilizado para identificar a região
- `Point2i points[4]` : conjunto de 4 pontos, em qualquer ordem, utilizados para delimitar o quadrilátero da região de entrada

---

`struct JVirtualLoopCalibRegion`

Descrição

Struct utilizada para definir a região de calibração da câmera. A região de calibração deve ser formada por um retângulo quando posicionado no plano do solo.

Membros

- `Point2i points[4]` : conjunto de 4 pontos, em qualquer ordem, utilizados para delimitar o quadrilátero da região de entrada
- `float width` : largura em metros da região
- `float height` : altura em metros da região

`struct JVirtualLoopAbandonObject`

Descrição

Struct utilizada para descrever a região ocupada por um objeto abandonado detectado.

Membros

- `uint32_t id` : ID único da região abandonada, este valor é consistente entre eventos.
- `uint32_t x` : coordenada X do ponto superior esquerdo do retângulo contendo a região abadonada
- `uint32_t y` : coordenada Y do ponto superior esquerdo do retângulo contendo a região abadonada
- `uint32_t width` : largura do rentângulo contendo a região abandonada
- `uint32_t height` : altura do retângulo contendo a região abandonada

`struct JVirtualLoopEvent`

Descrição

Struct utilizada para encapsular um evento ocorrido em uma **ActiveRegion**.

Membros

- `JVirtualLoopEntryRegion origin` : **EntryRegion** onde o evento foi originado
- `JVirtualLoopActiveRegion target` : **ActiveRegion** onde o evento foi detectado
- `JTimestamp timestamp` : timestamp do evento relativo ao servidor
- `JRegionState state` : estado da **ActiveRegion** no momento do evento
- `void* extra` : campo utilizado para passar atributos extras do evento. Ver **enum JEventExtraField**.

`struct JVirtualLoopLicenseInfo`

Descrição

Struct utilizada para armazenar as informações sobre a licença utilizada pela biblioteca.

Membros

- `uint64_t serial` : serial number da licença
- `char customer[64]` : nome do cliente que adquiriu a licença
- `int maxThreads` : número máximo de threads de processamento habilitadas
- `int maxConnections` : número máximo de conexões paralelas habilitadas
- `int state` : estado da licença - ver **Códigos de retorno de função**
- `int ttl` : time-to-live em horas para licenças do tipo RTC. Este campo possui o valor **-1** caso a licença não seja expirável

---

`struct JVirtualLoopServerInfo`

Descrição

Struct utilizada para armazenar informações de licença e versão de um servidor Virtual Loop.

Membros

- `JVirtualLoopLicenseInfo license` : estrutura contendo informações sobre a licença do servidor - ver **struct JVirtualLoopLicenseInfo**
- `int major` : valor do major da versão da biblioteca utilizada pelo servidor
- `int minor` : valor do minor da versão da biblioteca utilizada pelo servidor
- `int release` : valor do release da versão da biblioteca utilizada pelo servidor
- `char sha1[64]` : valor do SHA1 da versão da biblioteca utilizada pelo servidor

---

`struct JVirtualLoopConfig`

Descrição

Struct C utilizada para configurar um cliente do VirtualLoop. Armazena os dados de conexão com o servidor e as callbacks que devem ser chamadas.

Membros

- `const char* mjpeg_url` : string contendo a URL do fluxo Mjpeg no formato `http://<IP>[:PORT]/[PATH]`
- `const char* server_ip` : endereço de IP do servidor no formato **A.B.C.D**
- `int32_t server_port` : porta TCP utilizada pelo servidor
- `int32_t frame_queue_cap` : tamanho máximo da fila de frames mantidos pelo servidor [valor recomendado: 100]
- `void* user_ctx` : ponteiro para uma estrutura de contexto fornecida pelo usuário. Este ponteiro é passado como argumento nas chamadas das callbacks.
- `void (PTXAPI *on_event_cb)` : ponteiro para uma callback a ser chamada quando o laço mudar de estado

- `void (PTXAPI *on_abandon_cb)` : ponteiro para um callback a ser chamada quando ocorrer a adição ou a remoção de um objeto na lista de objetos abandonados
- `void (PTXAPI *on_error_cb)` : ponteiro para uma callback a ser chamada quando ocorrer um erro
- `void (PTXAPI *on_frame_cb)` : ponteiro para uma callback a ser chamada quando uma requisição **jl_vl_request_last_frame** completar

> **Argumentos:** `on_event_cb`
>
> - `const JVirtualLoopEvent* event` : ponteiro para a struct contendo os dados do evento
> - `const uint8_t* jpeg_ptr` : ponteiro para o frame em formato JPEG do evento, válido apenas durante a chamada da callback
> - `uint32_t jpeg_size` : tamanho do JPEG em bytes
> - `void* user_ctx` : argumento opcional fornecido pelo usuário no campo `void* user_ctx`
>
> **Argumentos:** `on_abandon_cb`
>
> - `const JVirtualLoopAbandonObject* regions` : ponteiro para um vetor contendo a lista objetos abandonados presentes
> - `uint32_t regions_size` : número de objetos abandonados presentes no vetor
> - `const uint8_t* jpeg_ptr` : ponteiro para o frame em formato JPEG do evento, válido apenas durante a chamada da callback
> - `uint32_t jpeg_size` : tamanho do JPEG em bytes
> - `void* user_ctx` :argumento opcional fornecido pelo usuário no campo `void* user_ctx`
>
> **Argumentos:** `on_error_cb`
>
> - `int32_t error_id` : código de identificação do erro - ver **Códigos de retorno de função**
> - `void* user_ctx` : argumento opcional fornecido pelo usuário no campo `void* user_ctx`
>
> **Argumentos:** `on_frame_cb`
>
> - `const uint8_t* jpeg_ptr` : ponteiro para o frame em formato JPEG, válido apenas durante a chamada da callback
> - `uint32_t jpeg_size` : tamanho do JPEG em bytes
> - `void* user_ctx` : argumento opcional fornecido pelo usuário no campo `void* user_ctx`

4.1.1.2. Métodos

`jl_vl_create_handle`

Protótipo da Função

```
PTXEXPORT JVirtualLoopHandle* PTXAPI jl_vl_create_handle();
```

Descrição

Função utilizada para criar um handle cliente VirtualLoop.

Parâmetros

Nenhum

Retorno

Ponteiro do tipo `JVirtualLoopHandle*` contendo o handle criado ou `NULL` em caso de erro

`jl_vl_destroy_handle`

Protótipo da Função

```
PTXEXPORT void PTXAPI jl_vl_destroy_handle(JVirtualLoopHandle* handle);
```

Descrição

> **Atenção:** essa função não pode ser chamada a partir de uma callback

Função utilizada para destruir um `handle` cliente. Caso o `handle` esteja conectado a um servidor, esta função chama internamente a função **jl_vl_disconnect** antes de destruir o handle.

Parâmetros

- `JVirtualLoopHandle* handle` : ponteiro válido para um handle cliente VirtualLoop

Retorno

Nenhum

---

jl_vl_clean_handle

Protótipo da Função

```
#define jl_vl_clean_handle(handle) { jl_vl_destroy_handle(handle); handle = NULL; }
```

Descrição

Macro utilizado para destruir um `handle` cliente e invalidar o ponteiro. É importante atribuir `NULL` ao ponteiro do handle após a destruição para garantir que as funções da API possam verificar se o ponteiro é válido antes de executar.

Parâmetros

- `JVirtualLoopHandle* handle` : ponteiro válido para um handle cliente VirtualLoop

Retorno

Nenhum

---

jl_vl_connect

Protótipo da Função

```
PTXEXPORT int PTXAPI jl_vl_connect(JVirtualLoopHandle* handle, JVirtualLoopConfig* config);
```

Descrição

Função utilizada para conectar um `handle` cliente a um servidor de VirtualLoop utilizando as configurações definidas em `config`.

Parâmetros

- `JVirtualLoopHandle* handle` : ponteiro válido para um handle cliente VirtualLoop
- `JVirtualLoopConfig* config` : ponteiro para um struct de configuração válida

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

jl_vl_connect_info

Protótipo da Função

```
PTXEXPORT int PTXAPI jl_vl_connect_info(JVirtualLoopHandle* handle, JVirtualLoopConfig* config, JVirtualLoopServerInfo* info
```

Descrição

Possui a mesma funcionalidade da função **jl_vl_connect** mas recebe um parâmetro adicional para receber informações sobre a licença e a versão do servidor.

Parâmetros

- `JVirtualLoopHandle* handle` : ponteiro válido para um handle cliente VirtualLoop
- `JVirtualLoopConfig* config` : ponteiro para um struct de configuração válida
- `JVirtualLoopServerInfo* info` : ponteiro para um struct que armazenará as informações de licença e a versão do servidor.

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

jl_vl_disconnect

Protótipo da Função

```
PTXEXPORT int PTXAPI jl_vl_disconnect(JVirtualLoopHandle* handle);
```

Descrição

> **Atenção:** essa função não pode ser chamada a partir de uma callback. Utilize a função **jl_vl_disconnect_cb** para este caso.

Função utilizada para desconectar um `handle` VirtualLoop. Esta função bloqueia até que a desconexão seja concluída e os eventos pendentes sejam executados.

Parâmetros

- `JVirtualLoopHandle* handle` : ponteiro válido para um handle cliente VirtualLoop

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

jl_vl_disconnect_cb

Protótipo da Função

```
PTXEXPORT int PTXAPI jl_vl_disconnect_cb(JVirtualLoopHandle* handle);
```

Descrição

> **Atenção:** essa função só pode ser chamada a partir de uma callback. Ver **jl_vl_disconnect**.

Função utilizada para desconectar um `handle` VirtualLoop de dentro de uma callback. Esta função retorna imediatamente.

Parâmetros

- `JVirtualLoopHandle* handle` : ponteiro válido para um handle cliente VirtualLoop

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

jl_vl_request_last_frame

Protótipo da Função

```
PTXEXPORT int PTXAPI jl_vl_request_last_frame(JVirtualLoopHandle* handle);
```

Descrição

Função utilizada para solicitar o último quadro de vídeo processado pelo servidor. Esta função é assíncrona, retorna imediatamente e só tem efeito se o handle já estiver conectado ao servidor. A callback `on_frame_cb` é chamada caso o quadro seja recebido com sucesso. Caso o servidor não tenha recebido nenhum quadro até o momento da requisição, a callback `on_error_cb` é chamada com o código `PTX_LAST_FRAME_UNAVAILABLE`.

Devido à fila de frames do servidor, o frame retornado pode ser relativo a um momento do passado, não havendo garantias de sincronismo com o fluxo Mjpeg. Além disso, chamadas consecutivas a esta função podem retornar um mesmo quadro caso nenhum novo quadro tenha sido recebido ou processado pelo servidor no intervalo entre as chamadas.

Parâmetros

- `JVirtualLoopHandle* handle` : ponteiro válido para um handle cliente VirtualLoop

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

jl_vl_set_calib_region

Protótipo da Função

```
PTXEXPORT int PTXAPI jl_vl_set_calib_region(JVirtualLoopHandle* handle, JVirtualLoopCalibRegion* cr);
```

Descrição

Função utilizada para definir a **CalibRegion** de uma instância do VirtualLoop. A definição desta região é obrigatória apenas nas aplicações que necessitem de medição de velocidade. Caso uma **CalibRegion** já tenha sido definida para um `handle`, está função retorna `PTX_API_CALL_HAS_NO_EFFECT`.

Ver **Demarcação da CalibRegion** para maiores informações.

Parâmetros

- `JVirtualLoopHandle* handle` : ponteiro válido para um handle cliente VirtualLoop
- `JVirtualLoopCalibRegion* cr` : ponteiro válido para uma **struct JVirtualLoopCalibRegion** contendo as configurações da **CalibRegion**

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

jl_vl_add_entry_region

Protótipo da Função

```
PTXEXPORT int PTXAPI jl_vl_add_entry_region(JVirtualLoopHandle* handle, JVirtualLoopEntryRegion* er)
```

Descrição

Função utilizada para adicionar uma **EntryRegion** a uma instância do VirtualLoop. Um `handle` só pode adicionar uma **EntryRegion** após estar conectado a um servidor VirtualLoop, caso contrário a chamada retorna com erro `PTX_API_CALL_HAS_NO_EFFECT`. Para redefinir uma região, deve-se destruir e recriar o handle.

> **Atenção:** o número de **EntryRegions** influencia diretamente na carga de processamento do servidor VirtualLoop

Parâmetros

- `JVirtualLoopHandle* handle` : ponteiro válido para um handle cliente VirtualLoop
- `JVirtualLoopEntryRegion* er` : ponteiro válido para uma **struct JVirtualLoopEntryRegion** contendo as configurações da **EntryRegion**

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

jl_vl_add_active_region

Protótipo da Função

```
PTXEXPORT int PTXAPI jl_vl_add_active_region(JVirtualLoopHandle* handle, JVirtualLoopActiveRegion* ar);
```

Descrição

Função utilizada para adicionar uma **ActiveRegion** a uma instância do VirtualLoop. Um `handle` só pode adicionar uma **EntryRegion** após estar conectado a um servidor VirtualLoop, caso contrário a chamada retorna com erro `PTX_API_CALL_HAS_NO_EFFECT`. Para redefinir uma região, deve-se destruir e recriar o handle.

> **Atenção:** o número de **ActiveRegion** tem pouco impacto na carga de processamento do servidor VirtualLoop

Parâmetros

- `JVirtualLoopHandle* handle` : ponteiro válido para um handle cliente VirtualLoop
- `JVirtualLoopActiveRegion* ar` : ponteiro válido para uma **struct JVirtualLoopActiveRegion** contendo as configurações da **ActiveRegion**

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

jl_vl_add_active_region

Protótipo da Função

```
PTXEXPORT int PTXAPI jl_vl_add_abandon_region(JVirtualLoopHandle* handle, JVirtualLoopAbandonDetRegion* abr);
```

Descrição

Função utilizada para adicionar uma **AbandonDetRegion** a uma instância do VirtualLoop. Um `handle` só pode adicionar uma **AbandonDetRegion** após estar conectado a um servidor VirtualLoop, caso contrário a chamada retorna com erro `PTX_API_CALL_HAS_NO_EFFECT`. Para redefinir uma região, deve-se destruir e recriar o handle.

> **Atenção:** o número de **AbandonDetRegion** impacto na carga de processamento do servidor VirtualLoop

Parâmetros

- `JVirtualLoopHandle* handle` : ponteiro válido para um handle cliente VirtualLoop
- `JVirtualLoopAbandonDetRegion* ar` : ponteiro válido para uma **struct JVirtualLoopAbandonDetRegion** contendo as configurações da **AbandonDetRegion**

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

jl_vl_set_float_property

Protótipo da Função

```
PTXEXPORT int PTXAPI jl_vl_set_float_property(JVirtualLoopHandle* handle, JVirtualLoopProperty prop, float value);
```

Descrição

Altera o valor de uma variável do tipo float da configuração.

Parâmetros

- `JVirtualLoopHandle* handle` : ponteiro válido para um handle cliente VirtualLoop
- `JVirtualLoopProperty prop` : enum que identifica qual a propriedade a ser alterada. Ver **enum JVirtualLoopProperty**
- `float value` : valor que deverá ser atribuido à propriedade

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

jl_vl_get_float_property

Protótipo da Função

```
PTXEXPORT int PTXAPI jl_vl_get_float_property(JVirtualLoopHandle* handle, JVirtualLoopProperty prop, float* value);
```

Descrição

Lê o valor de uma variável do tipo float da configuração.

Parâmetros

- `JVirtualLoopHandle* handle` : ponteiro válido para um handle cliente VirtualLoop
- `JVirtualLoopProperty prop` : enum que identifica qual a propriedade a ser lida. Ver **enum JVirtualLoopProperty**
- `float* value` : ponteiro para variável float onde será escrito o valor da propriedade

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

jl_vl_enable_detection

Protótipo da Função

```
PTXEXPORT int PTXAPI jl_vl_enable_detection(JVirtualLoopHandle* handle);
```

Descrição

Esta função deve ser chamada após a configuração de todas as regiões de uma instância do VirtualLoop: `CalibRegion`, `EntryRegion`, `ActiveRegion` e `AbandonDetRegion`. Ela serve para indicar ao servidor que todas as configurações já foram feitas e a detecção pode começar. Nenhuma região pode ser adicionada após a chamada dessa função, sendo necessário destruir e recriar o handle caso seja necessário fazer alterações.

Veja o arquivo `VirtualLoopClientSample.cpp` para um exemplo de uso desta função.

Parâmetros

- `JVirtualLoopHandle* handle` : ponteiro válido para um handle cliente VirtualLoop

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

jl_vl_get_handle_status

Protótipo da Função

```
PTXAPI int jl_vl_get_handle_status (
    JVirtualLoopHandle* handle,
    JVirtualLoopHandleStatus* status
);
```

Descrição

Função utilizada para recuperar o estado do handle cliente do VirtualLoop através de em uma **struct JVirtualLoopHandleStatus**.

Parâmetros

- `JVirtualLoopHandle* handle` : ponteiro válido para um handle cliente VirtualLoop
- `JVirtualLoopHandleStatus* status` : ponteiro válido para uma **struct JVirtualLoopHandleStatus**

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

jl_vl_has_event_extra

Protótipo da Função

```
int jl_vl_has_event_extra(
    const JVirtualLoopEvent* event,
    JEventExtraField field
);
```

Descrição

Função utilizada para checar se um determinado valor está presente no campo `void* extra` de um `JVirtualLoopEvent`.

Parâmetros

- `const JVirtualLoopEvent* event` : ponteiro para um evento
- `JEventExtraField field` : nome do campo extra. Ver **enum JEventExtraField** para uma descrição dos possíveis campos

Retorno

- `PTX_SUCCESS` se o campo existir, caso contrário `PTX_INVALID_PARAMETER`

---

jl_vl_get_event_extra_float

Protótipo da Função

```
float jl_vl_get_event_extra_float(
    const JVirtualLoopEvent* event,
    JEventExtraField field,
    float dvalue
);
```

Descrição

Função utilizada para ler um determinado valor do tipo `float` presente no campo `void* extra` de um `JVirtualLoopEvent`.

Parâmetros

- `const JVirtualLoopEvent* event` : ponteiro para um evento
- `JEventExtraField field` : nome do campo extra. Ver **enum JEventExtraField** para uma descrição dos possíveis campos
- `float dvalue` : valor default a ser retornado caso o campo não exista

Retorno

- Valor do campo caso existente ou `dvalue`

---

jl_vl_get_version

Protótipo da Função

```
int jl_vl_get_version(int* major, int* minor, int* release);
```

Descrição

Função utilizada para ler a versão da biblioteca do VirtualLoop

Parâmetros

- `int* major` : ponteiro para um inteiro onde será armazenado o major da versão
- `int* minor` : ponteiro para um inteiro onde será armazenado o minor da versão
- `int* release` : ponteiro para um inteiro onde será armazenado o release da versão

Retorno

- `PTX_SUCCESS`

---

jl_vl_get_build_SHA1

Protótipo da Função

```
const char* jl_vl_get_build_SHA1();
```

Descrição

Função utilizada para ler o SHA1 do release da biblioteca

Parâmetros

Nenhum

Retorno

- String C encodada em ASCII contendo o SHA1 do release da biblioteca

---

jl_vl_load_session_file

Protótipo da Função

```
int jl_vl_load_session_file(JVirtualLoopHandle* handle, const char* filename);
```

Descrição

Função utilizada para carregar as configurações dos laços à partir de um arquivo de sessão (*session file*) do VirtualLoop. Um *session file* é um arquivo JSON cujo formato é apresentado a seguir.

```
{
    "jidosha-light" : {
        "header" : {
            "er_cnt" : 1,
            "ar_cnt" : 2,
            "abr_cnt" : 0
        },
        "data" : {
            "cbw" : 1.0,
            "cbh" : 2.0,
            "cb"  : [x0,y0,x1,y1,x2,y2,x3,y3],
            "er0" : [x0,y0,x1,y1,x2,y2,x3,y3],
            "ar0" : [x0,y0,x1,y1,x2,y2,x3,y3],
            "ar1" : [x0,y0,x1,y1,x2,y2,x3,y3]
        },
        "help" : {
            "desc"  : "Este dicionário é fornecido apenas como especificação do formato do session file",
            "note"  : "Todos os pontos das reigões devem ser inseridos no formato [x0,y0,...,x3,y3]",
            "cb"    : "4 pontos da CalibRegion. Se este campo for omitido a CalibRegion é ignorada",
            "cbw"   : "Largura em metros da CalibRegion",
            "cbh"   : "Altura em metros da CalibRegion",
            "erN"   : "Pontos da n-ésima EntryRegion no intervalo [0..N-1]",
            "arN"   : "Pontos da n-ésima ActiveRegion no intervalo [0..N-1]",
            "abrN"  : "Pontos da n-ésima AbandonDetRegion no intervalo [0..N-1]",
            "er_cnt" : "Número de EntryRegions dentro de data. O parser procura pelo formato: er%d",
            "ar_cnt" : "Número de ActiveRegions dentro de data. O parser procura pelo formato: ar%d",
            "ar_cnt" : "Número de AbandonDetRegions dentro de data. O parser procura pelo formato: ar%d",
            "ar_df" : "ActiveRegion deb_frames",
            "ar_sf" : "ActiveRegion stop_frames",
            "ar_st" : "ActiveRegion sense_thresh"
        }
    }

}
```

Parâmetros

- `JVirtualLoopHandle* handle` : ponteiro para um inteiro onde será armazenado o major da versão
- `const char* filename` : caminho para o arquivo de configuração

Retorno

- `PTX_SUCCESS` caso as configurações da sessão sejam carregadas com sucesso ou um código de erro - ver **Códigos de retorno de função**

4.1.2. API Servidor

```
/* vl_api_server.h */

//================================================================================
// INCLUDES
//================================================================================
#include "ptx/ptx_common.h"

//================================================================================
// TYPES
//================================================================================
typedef struct JVirtualLoopServer JVirtualLoopServer;

//================================================================================
// METHODS
//================================================================================
/* server */
PTXAPI JVirtualLoopServer* jl_vl_create_server(int port, int conns);
PTXAPI void jl_vl_destroy_server(JVirtualLoopServer* server);
#define jl_vl_clean_server(server) { jl_vl_destroy_server(handle); server = NULL; }
```

4.1.2.1. Tipos

`struct JVirtualLoopServer`

Descrição

Tipo opaco utilizado para armazenar o handle de um servidor VirtualLoop.

Membros

Nenhum

4.1.2.2. Métodos

`jl_vl_create_server`

Protótipo da Função

```
PTXAPI JVirtualLoopServer* jl_vl_create_server(int port, int conns);
```

Descrição

Função utilizada para criar um e inicializar servidor VirtualLoop.

Parâmetros

- `int port`: valor da porta TCP utilizada pelo servidor
- `int conns`: número máximo de conexões simultâneas que o servidor pode aceitar (o valor máximo é limitado pela licença)

Retorno

Ponteiro do tipo `JVirtualLoopServer*` contendo o handle criado ou `NULL` em caso de erro. Essa função falha quando o socket do servidor não consegue fazer bind na porta especificada ou não há memória livre suficiente.

`jl_vl_destroy_server`

Protótipo da Função

```
PTXAPI void jl_vl_destroy_server(JVirtualLoopServer* server);
```

Descrição

Função utilizada para interromper e destruir um servidor VirtualLoop.

Parâmetros

- `JVirtualLoopServer* server`: ponteiro válido para um handle servidor VirtualLoop

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

`jl_vl_clean_server`

Protótipo da Função

```
#define jl_vl_clean_server(server) { jl_vl_destroy_server(handle); server = NULL; }
```

Descrição

Macro utilizado para destruir um handle servidor VirtualLoop e invalidar o ponteiro do handle. É importante atribuir `NULL` ao ponteiro do handle após a destruição para garantir que um handle inválido não seja passado para as funções.

Parâmetros

- `JVirtualLoopServer* server`: ponteiro válido para um handle servidor VirtualLoop

Retorno

Nenhum

4.1.3. Códigos de retorno de função

Os código de retorno de função são compartilhados entre as bibliotecas desenvolvidas pela **Pumatronix Equipamentos Eletrônicos** e estão definidos no header `ptx/common/ptx_codes.h`. Os códigos relevantes para o **VirtualLoop** são os seguintes:

- `PTX_SUCCESS` : Ação retornou com sucesso
- `PTX_INVALID_IMAGE_TYPE` : Tipo inválido de imagem fornecido
- `PTX_INVALID_PARAMETER` : Parâmetro inválido fornecido
- `PTX_API_CALL_NOT_SUPPORTED` : Chamada não suportada pela biblioteca para a plataforma
- `PTX_INVALID_ROI` : Os pontos fornecidos para a ROI são inválidos
- `PTX_INVALID_HANDLE` : O ponteiro para o handle fornecido é inválido
- `PTX_API_CALL_HAS_NO_EFFECT` : A chamada não teve efeito `no-op` - algumas chamadas possuem precedência, veja a documentação do método
- `PTX_LICENSE_INVALID` : A fonte de licença é inválida
- `PTX_LICENSE_EXPIRED` : A fonte de licença expirou
- `PTX_LICENSE_MAX_THREADS_EXCEEDED` : O número máximo de threads de processamento foi excedido
- `PTX_LICENSE_UNTRUSTED_RTC` : A fonte de tempo do sistema não é confiável (licença RTC)
- `PTX_LICENSE_MAX_CONNS_EXCEEDED` : O número máximo de conexões de processamento foi excedido
- `PTX_LICENSE_UNAUTHORIZED_PRODUCT` : O produto não está autorizado pela licença
- `PTX_CONNECT_FAILED` : Não foi possível concluir a conexão com o servidor
- `PTX_SOCKET_DISCONNECT` : A conexão com o servidor foi interrompida de maneira forçada
- `PTX_SOCKET_QUEUE_TIMEOUT` : Timeout não fila de requisições da conexão
- `PTX_SOCKET_QUEUE_FULL` : Fila de requisições cheia
- `PTX_SOCKET_IO_ERROR` : Erro de escrita ou leitura do socket
- `PTX_SOCKET_WRITE_FAILED` : Erro de escrita do socket
- `PTX_SOCKET_READ_TIMEOUT` : Tempo máximo para a recepção excedido
- `PTX_INVALID_RESPONSE` : Resposta recebida veio falformada
- `PTX_HANDLE_QUEUE_FULL` : File de requisições do handle está cheia
- `PTX_INVALID_REQUEST` : Requisição inválida
- `PTX_INVALID_MESSAGE` : Mensagem de requisição inválida
- `PTX_INVALID_STREAM_FRAME` : Frame inválido
- `PTX_FRAME_QUEUE_FULL` : File de frames cheia
- `PTX_LAST_FRAME_UNAVAILABLE` : Frame indisponível no momento
- `PTX_SERVER_CONN_LIMIT_REACHED` : Número máximo de conexões ao servidor excedido
- `PTX_MJPEG_ERROR_BASE` : Offset para os erros relativos ao Mjpeg (este valor nunca é utilizado como erro)
- `PTX_MJPEG_HTTP_HEADER_OVERFLOW` : O cabeçalho da resposta à requisição HTTP do Mjpeg veio corrompido
- `PTX_MJPEG_HTTP_RESPONSE_NOT_OK` : O servidor Mjpeg não aceitou a requisição HTTP
- `PTX_MJPEG_HTTP_CONTENT_TYPE_ERROR` : O servidor Mjpeg respondeu a requisição com `content-type` inválido
- `PTX_MJPEG_HTTP_CONTENT_LENGTH_ERROR` : O servidor Mjpeg respondeu a requisição com `content-length` inválido
- `PTX_MJPEG_HTTP_FRAME_BOUNDARY_NOT_FOUND` : O servidor Mjpeg não adicionou corretamente o `frame-boundary` ao quadro
- `PTX_MJPEG_CONNECTION_CLOSED` : A conexão foi encerrada pelo servidor Mjpeg
- `PTX_MJPEG_CONNECT_FAILED` : Não foi possível conectar ao servidor Mjpeg
- `PTX_UNKNOWN_ERROR` : Erro desconhecido, contate **contato@pumatronix.com.br**

4.2. API VirtualLoop DELPHI

A API Delphi está disponível apenas na versão Windows do SDK.

4.2.1 API Cliente

```
unit vl_client;

interface

//=============================================================================
// API Constants
//=============================================================================
const
    //API CALL RETURN CODES
    //SUCCESS
    PTX_SUCCESS                             = 0;

    //BASIC ERRORS
    PTX_FILE_NOT_FOUND                      = 1;
    PTX_INVALID_IMAGE                       = 2;
    PTX_INVALID_IMAGE_TYPE                  = 3;
    PTX_INVALID_PARAMETER                   = 4;
    PTX_COUNTRY_NOT_SUPPORTED               = 5;
    PTX_API_CALL_NOT_SUPPORTED              = 6;
    PTX_INVALID_ROI                         = 7;
    PTX_INVALID_HANDLE                      = 8;
    PTX_API_CALL_HAS_NO_EFFECT              = 9;

    //LICENSE ERRORS
    PTX_LICENSE_INVALID                     = 16;
    PTX_LICENSE_EXPIRED                     = 17;
    PTX_LICENSE_MAX_THREADS_EXCEEDED        = 18;

    //NETWORK ERRORS
    PTX_CONNECT_FAILED                      = 200;
    PTX_SOCKET_DISCONNECT                   = 201;
    PTX_SOCKET_QUEUE_TIMEOUT                = 202;
    PTX_SOCKET_QUEUE_FULL                   = 203;
    PTX_SOCKET_IO_ERROR                     = 204;
    PTX_SOCKET_WRITE_FAILED                 = 205;
    PTX_SOCKET_READ_TIMEOUT                 = 206;
    PTX_INVALID_REQUEST                     = 207;
    PTX_INVALID_MESSAGE                     = 208;
    PTX_INVALID_STREAM_FRAME                = 209;
    PTX_INVALID_RESPONSE                    = 210;
    PTX_FRAME_QUEUE_FULL                    = 211;
    PTX_LAST_FRAME_UNAVAILABLE              = 212;
    PTX_SERVER_CONN_LIMIT_REACHED           = 213;

    //MJPEG ERRORS
    PTX_MJPEG_ERROR_BASE                    = 1000;
    PTX_MJPEG_HTTP_HEADER_OVERFLOW          = 1001;
    PTX_MJPEG_HTTP_RESPONSE_NOT_OK          = 1002;
    PTX_MJPEG_HTTP_CONTENT_TYPE_ERROR       = 1003;
    PTX_MJPEG_HTTP_CONTENT_LENGTH_ERROR     = 1004;
    PTX_MJPEG_HTTP_FRAME_BOUNDARY_NOT_FOUND = 1005;
    PTX_MJPEG_CONNECTION_CLOSED             = 1006;
    PTX_MJPEG_CONNECT_FAILED                = 1007;

    // OTHERS
    PTX_UNKNOWN_ERROR                       = 99999;

    // enum JRegionState
    PTX_REGION_STATE_IDLE                   = 0;
    PTX_REGION_STATE_ACTIVE                 = 1;
    PTX_REGION_STATE_HOLDING                = 2;
    PTX_REGION_STATE_LINE_STOP              = 3;
    PTX_REGION_INVALID                      = 4;

    // enum JEventExtraField
    PTX_EVENT_SPEED                         = 0;
    PTX_EVENT_COUNTER_TOTAL                 = 1;
    PTX_EVENT_COUNTER_SAME_ORIGIN           = 2;
    PTX_EVENT_VEHICLE_TYPE                  = 3;
    PTX_EVENT_COUNTER_DELTA                 = 4;

    // enum JVehicleType
    PTX_VEHICLE_TYPE_UNKNOWN                = 0;
    PTX_VEHICLE_TYPE_CAR                    = 1;
    PTX_VEHICLE_TYPE_MOTORCYCLE             = 2;
    PTX_VEHICLE_TYPE_TRUCK                  = 3;
    PTX_VEHICLE_TYPE_BUS                    = 4;

    // enum JVirtualLoopProperty
    PTX_PROPERTY_VEHICLE_TYPE_ENABLED       = 0;
    PTX_PROPERTY_AUTHORIZED_VEHICLE_TYPES   = 1;
    PTX_PROPERTY_ACTIVE_REGION_TIMEOUT      = 2;

//=============================================================================
// API Types
//=============================================================================
type

  // Helper Types
  Point2i = record
    x : Int32;
    y : Int32;
  end;
  Point2iVec4 = Array [0..3] of Point2i;
```

```
JTimestamp = record
  year  : UInt16;
  month : UInt16;
  day   : UInt16;
  hour  : UInt16;
  min   : UInt16;
  sec   : UInt16;
  msec  : UInt16;
  pad   : UInt16;
end;

// API Main Types
// struct JVirtualLoopHandle
JVirtualLoopHandle = Pointer;

// struct JVirtualLoopHandleStatus
JVirtualLoopHandleStatus = record
  is_connected : Integer;
end;
PJVirtualLoopHandleStatus = ^JVirtualLoopHandleStatus;

// struct JVirtualLoopEntryRegion
JVirtualLoopEntryRegion = record
  id      : UInt32;
  points  : Point2iVec4;
end;
PJVirtualLoopEntryRegion = ^JVirtualLoopEntryRegion;

// struct JVirtualLoopActiveRegion
JVirtualLoopActiveRegion = record
  id          : UInt32;
  points      : Point2iVec4;
  deb_frames  : UInt32;
  stop_frames : UInt32;
  sense_thresh : UInt32;
end;
PJVirtualLoopActiveRegion = ^JVirtualLoopActiveRegion;

// struct JVirtualLoopAbandonDetRegion
JVirtualLoopAbandonDetRegion = record
  id      : UInt32;
  points  : Point2iVec4;
end;
PJVirtualLoopAbandonDetRegion = ^JVirtualLoopAbandonDetRegion;

// struct JVirtualLoopCalibRegion
JVirtualLoopCalibRegion = record
  points  : Point2iVec4;
  width   : Single;
  height  : Single;
end;
PJVirtualLoopCalibRegion = ^JVirtualLoopCalibRegion;

// struct JVirtualLoopAbandonObject
JVirtualLoopAbandonObject = record
  id    : UInt32;
  x     : UInt32;
  y     : UInt32;
  width : UInt32;
  height : UInt32;
end;
PJVirtualLoopAbandonObject = ^JVirtualLoopAbandonObject;

// struct JVirtualLoopEvent
JVirtualLoopEvent = record
  origin    : JVirtualLoopEntryRegion;
  target    : JVirtualLoopActiveRegion;
  timestamp : JTimestamp;
  state     : Integer;
  extra     : Pointer;
end;
PJVirtualLoopEvent = ^JVirtualLoopEvent;

// Callback Pointer Types
// void (PTXAPI *on_event_cb)
TOnEventCb = procedure(
  event     : PJVirtualLoopEvent;
  jpeg_ptr  : PByte;
  jpeg_size : UInt32;
  user_ctx  : Pointer);
  stdcall;

// void (PTXAPI *on_abandon_cb)
TOnAbandonCb = procedure(
  regions       : PJVirtualLoopAbandonObject;
  regions_size  : UInt32;
  jpeg_ptr      : PByte;
  jpeg_size     : UInt32;
  user_ctx      : Pointer);
  stdcall;

// void (PTXAPI *on_error_cb)
TOnErrorCb = procedure(
  error_id : Int32;
  user_ctx : Pointer);
  stdcall;

// void (PTXAPI *on_frame_cb)
TOnFrameCb = procedure(
  jpeg_ptr  : PByte;
  jpeg_size : UInt32;
  user_ctx  : Pointer);
  stdcall;

// struct JVirtualLoopConfig
JVirtualLoopConfig = record
```

```
    mjpeg_url       : PAnsiChar;
    server_ip       : PAnsiChar;
    server_port     : Int32;
    frame_queue_cap : Int32;
    user_ctx        : Pointer;
    on_event_cb     : TOnEventCb;
    on_abandon_cb   : TOnAbandonCb;
    on_error_cb     : TOnErrorCb;
    on_frame_cb     : TOnFrameCb;
  end;
  PJVirtualLoopConfig = ^JVirtualLoopConfig;

  // struct JVirtualLoopLicenseInfo
  JVirtualLoopLicenseInfo = record
    serial          : UInt64;
    customer        : array [0..63] of AnsiChar;
    maxThreads      : Int32;
    maxConnections  : Int32;
    state           : Int32;
    ttl             : Int32;
  end;
  PJVirtualLoopLicenseInfo = ^JVirtualLoopLicenseInfo;

  // struct JVirtualLoopServerInfo
  JVirtualLoopServerInfo = record
    license : JVirtualLoopLicenseInfo;
    major   : Int32;
    minor   : Int32;
    release : Int32;
    sha1    : array [0..63] of AnsiChar;
  end;
  PJVirtualLoopServerInfo = ^JVirtualLoopServerInfo;

//=============================================================================
// API Functions
//=============================================================================
// handle
// create/destroy
function jl_vl_create_handle : JVirtualLoopHandle;
  stdcall; external 'libptx_virtualLoop.dll';

procedure jl_vl_destroy_handle(
  handle : JVirtualLoopHandle);
  stdcall; external 'libptx_virtualLoop.dll';

// requests
function jl_vl_connect(
  handle : JVirtualLoopHandle;
  config : PJVirtualLoopConfig) : Integer;
  stdcall; external 'libptx_virtualLoop.dll';

function jl_vl_connect_info(
  handle : JVirtualLoopHandle;
  config : PJVirtualLoopConfig;
  info: PJVirtualLoopServerInfo) : Integer;
  stdcall; external 'libptx_virtualLoop.dll';

function jl_vl_disconnect(
  handle : JVirtualLoopHandle) : Integer;
  stdcall; external 'libptx_virtualLoop.dll';

function jl_vl_disconnect_cb(
  handle : JVirtualLoopHandle) : Integer;
  stdcall; external 'libptx_virtualLoop.dll';

function jl_vl_request_last_frame(
  handle : JVirtualLoopHandle) : Integer;
  stdcall; external 'libptx_virtualLoop.dll';

// regions
function jl_vl_set_calib_region(
  handle : JVirtualLoopHandle;
  cr : PJVirtualLoopCalibRegion) : Integer;
  stdcall; external 'libptx_virtualLoop.dll';

function jl_vl_add_entry_region(
  handle : JVirtualLoopHandle;
  er : PJVirtualLoopEntryRegion) : Integer;
  stdcall; external 'libptx_virtualLoop.dll';

function jl_vl_add_active_region(
  handle : JVirtualLoopHandle;
  ar : PJVirtualLoopActiveRegion) : Integer;
  stdcall; external 'libptx_virtualLoop.dll';

function jl_vl_add_abandon_region(
  handle : JVirtualLoopHandle;
  abr : PJVirtualLoopAbandonDetRegion) : Integer;
  stdcall; external 'libptx_virtualLoop.dll';

function jl_vl_set_float_property(
  handle : JVirtualLoopHandle;
  prop : Integer;
  value : Single) : Integer;
  stdcall; external 'libptx_virtualLoop.dll';

function jl_vl_get_float_property(
  handle : JVirtualLoopHandle;
  prop : Integer;
  value : PSingle) : Integer;
  stdcall; external 'libptx_virtualLoop.dll';

function jl_vl_enable_detection(
  handle : JVirtualLoopHandle) : Integer;
  stdcall; external 'libptx_virtualLoop.dll';

// status
```

```
  function jl_vl_get_handle_status(
    handle : JVirtualLoopHandle;
    status : PJVirtualLoopHandleStatus) : Integer;
    stdcall; external 'libptx_virtualLoop.dll';

  // events
  function jl_vl_has_event_extra(
    event : PJVirtualLoopEvent;
    field : Integer) : Integer;
    stdcall; external 'libptx_virtualLoop.dll';

  function jl_vl_get_event_extra_float(
    event : PJVirtualLoopEvent;
    field : Integer;
    dvalue : Single) : Single;
    stdcall; external 'libptx_virtualLoop.dll';

  // build info
  function jl_vl_get_version(
    major : PInteger;
    minor : PInteger;
    release : PInteger) : Integer;
    stdcall; external 'libptx_virtualLoop.dll';

  function jl_vl_get_build_SHA1 : PAnsiChar;
    stdcall; external 'libptx_virtualLoop.dll';

  // session file
  function jl_vl_load_session_file(
    handle    : JVirtualLoopHandle;
    filename  : PAnsiChar) : Integer;
    stdcall; external 'libptx_virtualLoop.dll';

  //============================================================================
  // Helper Functions
  //============================================================================
  function makePoint2i(x : Int32; y : Int32) : Point2i;

  //============================================================================
  // Implementation
  //============================================================================
  implementation

  function makePoint2i(x : Int32; y : Int32) : Point2i;
  var
    p : Point2i;
  begin
    p.x := x;
    p.y := y;
    result := p;
  end;

  end.
```

4.2.1.1. Tipos

`record Point2i`

Descrição

Ver **struct Point2i**

Membros

- `x : Int32`
- `y : Int32`

`Array Point2iVec4`

Descrição

Tipo auxiliar uilizado para definir uma `Array` contendo 4 membros do tipo **record Point2i**.

Membros

- `Point2i[0..3]`

`record JTimestamp`

Descrição

Ver **struct JTimestamp**

Membros

- `year : UInt16`
- `month : UInt16`

- `day : UInt16`
- `hour : UInt16`
- `min : UInt16`
- `sec : UInt16`
- `msec : UInt16`
- `pad : UInt16`

`type JVirtualLoopHandle`

Descrição

Ver **struct JVirtualLoopHandle**

Membros

Nenhum

`record JVirtualLoopHandleStatus`

Descrição

Ver **struct JVirtualLoopHandleStatus**

Membros

- `is_connected : Integer`

`record JVirtualLoopEntryRegion`

Descrição

Ver **struct JVirtualLoopEntryRegion**

Membros

- `id : UInt32`
- `points : Point2iVec4`

`record JVirtualLoopActiveRegion`

Descrição

Ver **struct JVirtualLoopActiveRegion**

Membros

- `id : UInt32`
- `points : Point2iVec4`
- `deb_frames : UInt32`
- `stop_frames : UInt32`
- `sense_thresh : UInt32`

`record JVirtualLoopAbandonDetRegion`

Descrição

Ver **struct JVirtualLoopAbandonDetRegion**

Membros

- `id : UInt32`
- `points : Point2iVec4`

`record JVirtualLoopCalibRegion`

Descrição

Ver **struct JVirtualLoopCalibRegion**

Membros

- `points : Point2iVec4`
- `width  : Single`
- `height : Single`

`record JVirtualLoopAbandonObject`

Descrição

Ver **struct JVirtualLoopAbandonObject**

Membros

- `id : UInt32`
- `x : UInt32`
- `y : UInt32`
- `width : UInt32`
- `height : UInt32`

`record JVirtualLoopEvent`

Descrição

Ver **struct JVirtualLoopEvent**

Membros

- `origin : JVirtualLoopEntryRegion`
- `target : JVirtualLoopActiveRegion`
- `timestamp : JTimestamp`
- `state : Integer`
- `extra : Pointer`

`type TOnEventCb`

Descrição

Tipo auxiliar utilizado para definir um ponteiro para uma função. Utilizado em **record JVirtualLoopConfig**.

Membros

- `event : PJVirtualLoopEvent`
- `jpeg_ptr : PByte`
- `jpeg_size : UInt32`
- `user_ctx : Pointer`

`type TOnAbandonCb`

Descrição

Tipo auxiliar utilizado para definir um ponteiro para uma função. Utilizado em **record JVirtualLoopConfig**.

Membros

- `regions : PJVirtualLoopAbandonObject`
- `regions_size : UInt32`
- `jpeg_ptr : PByte`
- `jpeg_size : UInt32`
- `user_ctx : Pointer`

`type TOnErrorCb`

Descrição

Tipo auxiliar utilizado para definir um ponteiro para uma função. Utilizado em **record JVirtualLoopConfig**.

Membros

- `error_id : Int32`
- `user_ctx : Pointer`

`type TOnFrameCb`

Descrição

Tipo auxiliar utilizado para definir um ponteiro para uma função. Utilizado em **record JVirtualLoopConfig**.

Membros

- `jpeg_ptr : PByte`
- `jpeg_size : UInt32`
- `user_ctx : Pointer`

`record JVirtualLoopConfig`

Descrição

Ver **struct JVirtualLoopConfig**

Membros

- `mjpeg_url : PAnsiChar`
- `server_ip : PAnsiChar`
- `server_port : Int32`
- `frame_queue_cap : Int32`
- `user_ctx : Pointer`
- `on_event_cb : TOnEventCb`
- `on_abandon_cb : TOnAbandonCb`
- `on_error_cb : TOnErrorCb`
- `on_frame_cb : TOnFrameCb`

4.2.1.2. Métodos

`jl_vl_create_handle`

Protótipo da Função

```
function jl_vl_create_handle : JVirtualLoopHandle;
```

Descrição

Ver método **jl_vl_create_handle** da API C.

Parâmetros

Nenhum

Retorno

Ponteiro do tipo `JVirtualLoopHandle` contendo o handle criado ou `Nil` em caso de erro

---

`jl_vl_destroy_handle`

Protótipo da Função

```
procedure jl_vl_destroy_handle(handle : JVirtualLoopHandle);
```

Descrição

Ver método **jl_vl_destroy_handle** da API C.

Parâmetros

Nenhum

Retorno

Nenhum

---

jl_vl_connect

Protótipo da Função

```
function jl_vl_connect(
    handle : JVirtualLoopHandle;
    config : PJVirtualLoopConfig) : Integer;
```

Descrição

Ver método **jl_vl_connect** da API C.

Parâmetros

- `handle : JVirtualLoopHandle`
- `config : PJVirtualLoopConfig`

Retorno

- `PTX_SUCCESS` em caso de sucesso ou um código de erro - ver **Códigos de retorno de função**

---

jl_vl_disconnect

Protótipo da Função

```
function jl_vl_connect(handle : JVirtualLoopHandle) : Integer;
```

Descrição

Ver método **jl_vl_disconnect** da API C.

Parâmetros

- `handle : JVirtualLoopHandle`

Retorno

- `PTX_SUCCESS` em caso de sucesso ou um código de erro - ver **Códigos de retorno de função**

---

jl_vl_disconnect_cb

Protótipo da Função

```
function jl_vl_disconnect_cb(handle : JVirtualLoopHandle) : Integer;
```

Descrição

Ver método **jl_vl_disconnect_cb** da API C.

Parâmetros

- `handle : JVirtualLoopHandle`

Retorno

- `PTX_SUCCESS` em caso de sucesso ou um código de erro - ver **Códigos de retorno de função**

---

jl_vl_request_last_frame

Protótipo da Função

```
function jl_vl_request_last_frame(handle : JVirtualLoopHandle) : Integer;
```

Descrição

Ver método **jl_vl_request_last_frame** da API C.

Parâmetros

- `handle : JVirtualLoopHandle`

Retorno

- `PTX_SUCCESS` em caso de sucesso ou um código de erro - ver **Códigos de retorno de função**

---

jl_vl_set_calib_region

Protótipo da Função

```
function jl_vl_set_calib_region(
  handle     : JVirtualLoopHandle;
  cr         : PJVirtualLoopCalibRegion) : Integer;
```

Descrição

Ver método **jl_vl_set_calib_region** da API C.

Parâmetros

- `handle : JVirtualLoopHandle`
- `cr : PJVirtualLoopCalibRegion`

Retorno

- `PTX_SUCCESS` em caso de sucesso ou um código de erro - ver **Códigos de retorno de função**

---

jl_vl_add_entry_region

Protótipo da Função

```
function jl_vl_add_entry_region(
  handle     : JVirtualLoopHandle;
  er         : PJVirtualLoopEntryRegion) : Integer;
```

Descrição

Ver método **jl_vl_add_entry_region** da API C.

Parâmetros

- `handle : JVirtualLoopHandle`
- `er : PJVirtualLoopEntryRegion`

Retorno

- `PTX_SUCCESS` em caso de sucesso ou um código de erro - ver **Códigos de retorno de função**

---

jl_vl_add_active_region

Protótipo da Função

```
function jl_vl_add_active_region(
  handle     : JVirtualLoopHandle;
  ar         : PJVirtualLoopActiveRegion) : Integer;
```

Descrição

Ver método **jl_vl_add_active_region** da API C.

Parâmetros

- `handle : JVirtualLoopHandle`
- `er : PJVirtualLoopEntryRegion`

Retorno

- `PTX_SUCCESS` em caso de sucesso ou um código de erro - ver **Códigos de retorno de função**

---

jl_vl_add_abandon_region

Protótipo da Função

```
function jl_vl_add_abandon_region(
  handle     : JVirtualLoopHandle;
  abr        : PJVirtualLoopAbandonDetRegion) : Integer;
```

Descrição

Ver método **jl_vl_add_abandon_region** da API C.

Parâmetros

- `handle : JVirtualLoopHandle`
- `abr : PJVirtualLoopAbandonDetRegion`

Retorno

- `PTX_SUCCESS` em caso de sucesso ou um código de erro - ver **Códigos de retorno de função**

---

jl_vl_enable_detection

Protótipo da Função

```
function jl_vl_enable_detection(
  handle : JVirtualLoopHandle) : Integer;
```

Descrição

Ver método **jl_vl_enable_detection** da API C.

Parâmetros

- `handle : JVirtualLoopHandle`

Retorno

- `PTX_SUCCESS` em caso de sucesso ou um código de erro - ver **Códigos de retorno de função**

---

jl_vl_get_handle_status

Protótipo da Função

```
function jl_vl_get_handle_status(
    handle : JVirtualLoopHandle;
    status : PJVirtualLoopHandleStatus) : Integer;
```

Descrição

Ver método **jl_vl_get_handle_status** da API C.

Parâmetros

- `handle`
- `status`

Retorno

- `PTX_SUCCESS` em caso de sucesso ou um código de erro - ver **Códigos de retorno de função**

---

jl_vl_has_event_extra

Protótipo da Função

```
function jl_vl_has_event_extra(
  event : PJVirtualLoopEvent;
  field : Integer) : Integer;
```

Descrição

Ver método **jl_vl_has_event_extra** da API C.

Parâmetros

- `event : PJVirtualLoopEvent`
- `field : Integer`

Retorno

Ver método **jl_vl_has_event_extra** da API C.

---

jl_vl_get_event_extra_float

Protótipo da Função

```
function jl_vl_get_event_extra_float(
  event : PJVirtualLoopEvent;
  field : Integer;
  dvalue : Single) : Single;
```

Descrição

Ver método **jl_vl_get_event_extra_float** da API C.

Parâmetros

- `event : PJVirtualLoopEvent`
- `field : Integer`
- `dvalue : Single`

Retorno

Ver método **jl_vl_get_event_extra_float** da API C.

---

jl_vl_get_version

Protótipo da Função

```
function jl_vl_get_version(
    major : PInteger;
    minor : PInteger;
    release : PInteger) : Integer;
```

Descrição

Ver método **jl_vl_get_version** da API C.

Parâmetros

- `major : PInteger`

- `minor : PInteger`
- `release : PInteger`

Retorno

- `PTX_SUCCESS`

---

jl_vl_get_build_SHA1

Protótipo da Função

```
function jl_vl_get_build_SHA1 : PAnsiChar;
```

Descrição

Ver método **jl_vl_get_build_SHA1** da API C.

Parâmetros

Nenhum

Retorno

- String contendo o SHA1 do release da biblioteca

---

jl_vl_load_session_file

Protótipo da Função

```
function jl_vl_load_session_file(
  handle    : JVirtualLoopHandle;
  filename  : PAnsiChar) : Integer;
```

Descrição

Ver método **jl_vl_load_session_file** da API C.

Parâmetros

- `handle : JVirtualLoopHandle`
- `filename : PAnsiChar`

Retorno

- `PTX_SUCCESS` em caso de sucesso ou um código de erro - ver **Códigos de retorno de função**

---

makePoint2i

Protótipo da Função

```
function makePoint2i(x : Int32; y : Int32) : Point2i;
```

Descrição

Função utilitária utilizada para criar um **record Point2i** a partir de dois números inteiros.

Parâmetros

- `x : Int32` : coordenada X do ponto
- `y : Int32` : coordenada Y do ponto

Retorno

- `Point2i`

4.3. API VirtualLoop Java

A API Java é wrapper da API C/C++ e está disponível para Linux PC/ARM e Windows.

4.3.1. API Comum

```
//=========================================================================
    // ENUMS
    //=========================================================================
    // JRegionState
    public static final int PTX_VL_REGION_STATE_IDLE         = 0;
    public static final int PTX_VL_REGION_STATE_ACTIVE       = 1;
    public static final int PTX_VL_REGION_STATE_HOLDING      = 2;
    public static final int PTX_VL_REGION_STATE_LINE_STOP    = 3;
    public static final int PTX_VL_REGION_STATE_INVALID      = 4;

    // JEventExtraField
    public static final int PTX_VL_EXTRA_FIELD_SPEED                  = 0;
    public static final int PTX_VL_EXTRA_FIELD_COUNTER_TOTAL          = 1;
    public static final int PTX_VL_EXTRA_FIELD_COUNTER_SAME_ORIGIN    = 2;
    public static final int PTX_VL_EXTRA_FIELD_VEHICLE_TYPE           = 3;
    public static final int PTX_VL_EXTRA_FIELD_COUNTER_DELTA          = 4;

    // JVehicleTypes
    public static final int  PTX_VL_VEHICLE_TYPE_UNKNOWN                 = 0;
    public static final int  PTX_VL_VEHICLE_TYPE_CAR                     = 1;
    public static final int  PTX_VL_VEHICLE_TYPE_MOTORCYCLE              = 2;
    public static final int  PTX_VL_VEHICLE_TYPE_TRUCK                   = 3;
    public static final int  PTX_VL_VEHICLE_TYPE_BUS                     = 4;

    // JVirtualLoopProperty
    public static final int  PTX_VL_PROPERTY_VEHICLE_TYPE_ENABLED      = 0;
    public static final int  PTX_VL_PROPERTY_AUTHORIZED_VEHICLE_TYPES  = 1;
    public static final int  PTX_VL_PROPERTY_ACTIVE_REGION_TIMEOUT     = 2;

    //=========================================================================
    // HELPER TYPES
    //=========================================================================
    // Point2i
    public static class Point2i {
        public int x;
        public int y;
        public Point2i(int x,int y) {
            this.x = x;
            this.y = y;
        }
        public String toString() {
            return "(" + String.valueOf(x) + "," + String.valueOf(y) + ")";
        }
    };
    // JTimestamp
    public static class Timestamp {
        public int year;
        public int month;
        public int day;
        public int hour;
        public int min;
        public int sec;
        public int msec;
        public Timestamp(int year, int month, int day, int hour, int min,
                         int sec, int msec) {
            this.year = year;
            this.month = month;
            this.day = day;
            this.hour = hour;
            this.min = min;
            this.sec = sec;
            this.msec = msec;
        }
    };
    // Version
    public static class Version {
        public int major;
        public int minor;
        public int release;
    };

    // Return values
    public static class MutableValue<T> {
        public T value;
    };

    //=========================================================================
    // MAIN TYPES
    //=========================================================================
    // AbstractRegion
    public static abstract class AbstractRegion {
        public final int id;
        public final List<Point2i> points;
        public AbstractRegion(int id, List<Point2i> points) throws IllegalArgumentException {
            if(points.size() != 4) {
                throw new IllegalArgumentException("points.size() != 4");
            }
            this.id = id;
            this.points = points;
        }
        public String toString() {
            return  points.get(0).toString() + "," +
                    points.get(1).toString() + "," +
                    points.get(2).toString() + "," +
                    points.get(3).toString();
```

```
        }
    };
    // JVirtualLoopEntryRegion
    public static class EntryRegion extends AbstractRegion {
        public EntryRegion(int id, List<Point2i> points) {
            super(id,points);
        }
    };
    // JVirtualLoopActiveRegion
    public static class ActiveRegion extends AbstractRegion {
        public final int debFrames;
        public final int stopFrames;
        public final int senseThresh;
        public ActiveRegion(int id, List<Point2i> points, int debFrames,
                            int stopFrames, int senseThresh) throws IllegalArgumentException {
            super(id,points);
            this.debFrames = debFrames;
            this.stopFrames = stopFrames;
            this.senseThresh = senseThresh;
        }
    };
    // JVirtualLoopAbandonDetRegion
    public static class AbandonDetRegion extends AbstractRegion {
        public AbandonDetRegion(int id, List<Point2i> points) {
            super(id,points);
        }
    };
    // JVirtualLoopCalibRegion
    public static class CalibRegion extends AbstractRegion {
        public final float widthInMeters;  /* width in meters of the rectangle */
        public final float heightInMeters; /* height in meters of the rectangle */
        public CalibRegion(List<Point2i> points, float widthInMeters, float heightInMeters) {
            super(0,points);
            this.widthInMeters = widthInMeters;
            this.heightInMeters = heightInMeters;
        }
    };
    // JVirtualLoopAbandonObject
    public static class AbandonObject {
        public final int id;
        public final int x;
        public final int y;
        public final int width;
        public final int height;
        AbandonObject(int id, int x, int y, int width, int height) {
            this.id = id;
            this.x = x;
            this.y = y;
            this.width = width;
            this.height = height;
        }

    };
    // JVirtualLoopEvent
    public static class Event {
        public EntryRegion origin;
        public ActiveRegion target;
        public Timestamp ts;
        public int state;
        public HashMap<Integer,Float> extra;
        Event(EntryRegion origin, ActiveRegion target, Timestamp ts, int state,
                HashMap<Integer,Float> extra) {
            this.origin = origin;
            this.target = target;
            this.ts = ts;
            this.state = state;
            this.extra = extra;
        }
    };

    public static class JVirtualLoopLicenseInfo {
        public String serial;
        public String customer;
        public int maxThreads;
        public int maxConnections;
        public int state;
        public int ttl;
    }

    public static class JVirtualLoopServerInfo {
        public JVirtualLoopLicenseInfo license;
        public Version version;
        public String sha1;
    }

    //========================================================================
    // METHODS
    //========================================================================
    public static native Version getVersion();
    public static native String getBuildSHA1();

    public static void loadLibrary() {
        System.loadLibrary("ptx_virtualLoopJava");
    }
```

4.3.1.1. Tipos

`JRegionState`

Descrição

Definição dos possíveis estados de uma **ActiveRegion**.

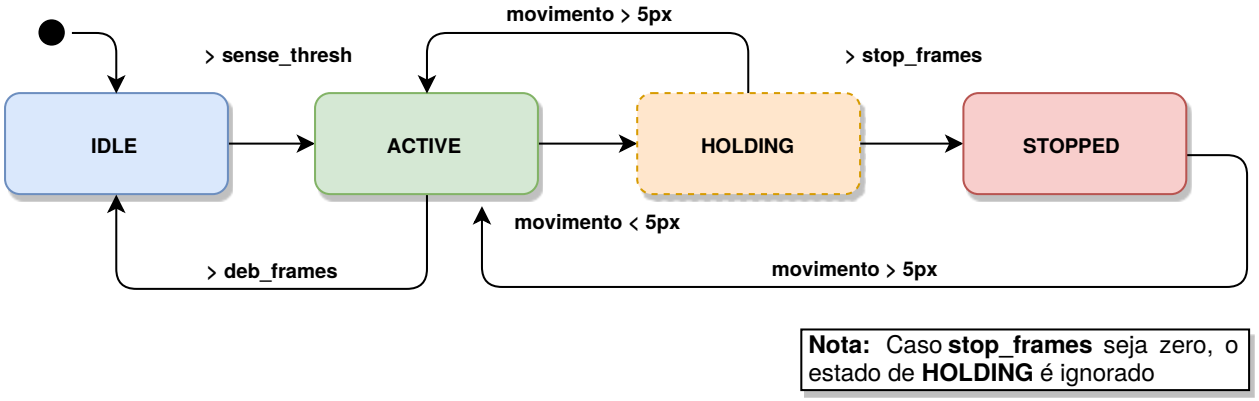


Diagrama de estados de uma ActiveRegion
*Para entender o significado dos parâmetros de transição ver **jl_vl_add_active_region***

Membros

- `PTX_VL_REGION_STATE_IDLE` : nenhum movimento detectado na região do laço
- `PTX_VL_REGION_STATE_ACTIVE` : movimento detectado na região do laço
- `PTX_VL_REGION_STATE_HOLDING` : parada detectada mas **stop_frames** ainda não excedido
- `PTX_VL_REGION_STATE_LINE_STOP` : parada detectada
- `PTX_VL_REGION_INVALID` : valor reservado para implementações futuras

JEventExtraField

Descrição

Definição dos possíveis campos adicionais da **struct JVirtualLoopEvent**.

Membros

- `PTX_VL_EVENT_SPEED` : velocidade média no momento do evento. Atualizada apenas nos eventos de `ACTIVE`
- `PTX_VL_EVENT_COUNTER_TOTAL` : valor acumulado de contagem. Atualizado nos eventos de `IDLE`
- `PTX_VL_EVENT_COUNTER_SAME_ORIGIN` : valor acumulado de contagem para eventos com a mesma origem do evento atual. Atualizado nos eventos de `IDLE`
- `PTX_VL_EVENT_VEHICLE_TYPE` : tipo do veículo que atravessa a região ativa - ver **enum JVehicleTypes**. Atualizada apenas nos eventos de `ACTIVE`

JVehicleTypes

Descrição

Definição das possíveis saídas de classificação dos veículos que passam por uma região ativa.

Membros

- `PTX_VL_VEHICLE_TYPE_UNKNOWN` : classe que define que o veículo não foi reconhecido
- `PTX_VL_VEHICLE_TYPE_CAR` : o veículo que atravessa a região ativa é do tipo carro
- `PTX_VL_VEHICLE_TYPE_MOTORCYCLE` : o veículo que atravessa a região ativa é do tipo moto
- `PTX_VL_VEHICLE_TYPE_TRUCK` : o veículo que atravessa a região ativa é do tipo caminhão
- `PTX_VL_VEHICLE_TYPE_BUS` : o veículo que atravessa a região ativa é do tipo ônibus

JVirtualLoopProperty

Descrição

Definição das possíveis configurações que podem ser realizadas no VirtualLoop.

Membros

- `PTX_VL_PROPERTY_VEHICLE_TYPE_ENABLED` : ativa ou desativa a classificação de tipo veicular (default: desativada)
- `PTX_VL_PROPERTY_AUTHORIZED_VEHICLE_TYPES` : Propriedade a ser utilizada como máscara para se ativar ou desativar determinada categoria de veículo. Uma categoria desativada não ativa o laço virtual e portanto não gera eventos. Os valores que habilitam cada categoria são da forma "0x1 << **JVehicleTypes**" e podem ser somados (default: 30 - todas as categorias habilitadas).

```
Exemplo:
```

```
16 (PTX_VL_VEHICLE_TYPE_BUS)
+8 (PTX_VL_VEHICLE_TYPE_TRUCK)
+4 (PTX_VL_VEHICLE_TYPE_MOTORCYCLE)
+2 (PTX_VL_VEHICLE_TYPE_CAR)
= 30 (todas as categorias habilitadas)
Nota: O valor da propriedade é um float e portanto não se deve utilizar operações bitwise.
```

- `PTX_VL_PROPERTY_ACTIVE_REGION_TIMEOUT` : número inteiro a ser fornecido em segundos que estabelece o tempo máximo de retenção do nível alto `STOPPED` da região ativa. Esse valor é estabelecido no momento de adição da região ativa ao cenário e não será atualizado em tempo real. Esse valor é global para todas as regiões ativas (default: 300s).

---

public static class Point2i

Descrição

Classe Java usada para representar um ponto 2D com coordenadas inteiras.

Membros

- `public int x` : coordenada x do ponto
- `public int y` : coordenada y do ponto

Métodos

toString

Protótipo da Função

```
public String toString();
```

Descrição

Conversor dos valores inteiros de coordenadas para string.

Parâmetros

Nenhum

Retorno

Retorna o valor das coordenadas do ponto em formato "(x,y)".

---

public static class Timestamp

Descrição

Classe Java usada para retornar um timestamp. O timestamp retornado pela callback de evento é relativo ao servidor e não tem garantia de ser monotônico.

Membros

- `public int year` : valor inteiro do ano [ex: 2017]
- `public int month` : valor inteiro do mês do ano [1-12]
- `public int day` : valor inteiro do dia do mês [1-31]
- `public int hour` : valor inteiro da hora do dia [0-23]
- `public int min` : valor inteiro do minuto [0-59]
- `public int sec` : valor inteiro do segundo [0-59]
- `public int msec` : valor inteiro do milisegundo [0-999]
- `public int pad` : preenchimento adicionado para garantir o alinhamento da struct em 32 bits [sempre 0]

---

public static class Version

Descrição

Tipo utilizado para armazenar a versão cliente do VirtualLoop.

Membros

- `public int major` : valor majoritário da versão

- `public int minor` : valor minoritário da versão
- `public int release` : valor da divulgação da versão

`public static class MutableValue`

Descrição

Tipo genérico utilizado para armazenar o retorno de funções do cliente do VirtualLoop.

Membros

- `public T value` : retorno genérico

`public static abstract class AbstractRegion`

Descrição

Classe abstrata responsável por representar os diferentes tipos de regiões do VirtualLoop (**EntryRegion**, **ActiveRegion**, **AbandonDetRegion**, **CalibRegion**).

Membros

- `public final int` : ID utilizado para identificar a região
- `public final List<Point2i>` : conjunto de 4 pontos, em qualquer ordem, utilizados para delimitar o quadrilátero da região de entrada

Métodos

`toString`

Protótipo da Função

```
    public String toString();
```

Descrição

Conversor dos valores inteiros de coordenadas dos vértices das regiões para string.

Parâmetros

Nenhum

Retorno

Retorna o valor das coordenadas da regiã0 em formato "(x0,y0),(x1,y1),(x2,y2),(x3,y3)".

`public static class EntryRegion`

Descrição

Classe utilizada para definir uma **EntryRegion**.

Membros

- `public final int id` : ID utilizado para identificar a região
- `public final List<Point2i>` : conjunto de 4 pontos, em qualquer ordem, utilizados para delimitar o quadrilátero da região de entrada

---

`public static class ActiveRegion`

Descrição

Classe utilizada para definir uma **ActiveRegion**.

Membros

- `public final int id` : ID utilizado para identificar a região
- `public final List<Point2i>` : conjunto de 4 pontos, em qualquer ordem, utilizados para delimitar o quadrilátero da região de ativação
- `public final int deb_frames` : número de frames utilizados para o debounce da região ativa. Valores típicos para este campo vão de **4** a **8**. Este parâmetro tem o intuito de reduzir multiplas detecções de veículos muito longos, como caminhões e ônibus

- `public final int stop_frames` : número de frames para que um evento de holding seja considerado uma parada. Este valor deve ser escolhido baseado na taxa de frames do fluxo Mjpeg
- `public final int sense_thresh` : valor de sensibilidade do laço no intervalo [1-30], sendo 1 mais sensível e 30 menos sensível. Este valor está relacionado a ocupação mínima do laço para que o movimento sejá considerado. Valores maiores devem ser utilizados para laços maiores. Aplicações noturnas ou com pouca iluminação devem utilizar valores menores

---

`public static class AbandonDetRegion`

Descrição

Classe utilizada para delimitar uma região de detecção de objectos abandonados.

Membros

- `public final int id` : ID utilizado para identificar a região
- `public final List<Point2i>` : conjunto de 4 pontos, em qualquer ordem, utilizados para delimitar o quadrilátero da região de detecção de objectos abandonados.

`public static class CalibRegion`

Descrição

Classe utilizada para definir a região de calibração da câmera. A região de calibração deve ser formada por um retângulo quando posicionado no plano do solo.

Membros

- `public final List<Point2i>` : conjunto de 4 pontos, em qualquer ordem, utilizados para delimitar o quadrilátero da região de entrada
- `public final float width` : largura em metros da região
- `public final float height` : altura em metros da região

`public static class AbandonObject`

Descrição

Classe utilizada para descrever a região ocupada por um objeto abandonado detectado.

Membros

- `public final int id` : ID único da região abandonada, este valor é consistente entre eventos.
- `public final int x` : coordenada X do ponto superior esquerdo do retângulo contendo a região abadonada
- `public final int y` : coordenada Y do ponto superior esquerdo do retângulo contendo a região abadonada
- `public final int width` : largura do rentângulo contendo a região abandonada
- `public final int height` : altura do retângulo contendo a região abandonada

`public static class Event`

Descrição

Class utilizada para encapsular um evento ocorrido em uma **ActiveRegion**.

Membros

- `public EntryRegion origin` : **EntryRegion** onde o evento foi originado
- `public ActiveRegion target` : **ActiveRegion** onde o evento foi detectado
- `public Timestamp timestamp` : timestamp do evento relativo ao servidor
- `public int state` : estado da **ActiveRegion** no momento do evento
- `public HashMap<Integer,Float> extra` : campo utilizado para passar atributos extras do evento. Ver **enum JEventExtraField**.

`public static class JVirtualLoopLicenseInfo`

Descrição

Classe utilizada para armazenar as informações sobre a licença utilizada pela biblioteca.

Membros

- `public String serial` : serial number da licença
- `public String customer[64]` : nome do cliente que adquiriu a licença
- `public int maxThreads` : número máximo de threads de processamento habilitadas
- `public int maxConnections` : número máximo de conexões paralelas habilitadas
- `public int state` : estado da licença - ver **Códigos de retorno de função**
- `public int ttl` : time-to-live em horas para licenças do tipo RTC. Este campo possui o valor **-1** caso a licença não seja expirável

---

`public static class JVirtualLoopServerInfo`

Descrição

Classe utilizada para armazenar informações de licença e versão de um servidor Virtual Loop.

Membros

- `public JVirtualLoopLicenseInfo license` : estrutura contendo informações sobre a licença do servidor - ver **public static class JVirtualLoopServerInfo**
- `public Version version` : valor da versão da biblioteca utilizada pelo servidor
- `public String sha1` : Md5sum correspondente ao build da biblioteca

4.3.1.2. Métodos

`getVersion`

Protótipo da Função

```
public static native Version getVersion();
```

Descrição

Função utilizada para se obter a versão do cliente VirtualLoop.

Parâmetros

Nenhum

Retorno

Objeto Version - ver **public static class Version**

`getBuildSHA1`

Protótipo da Função

```
public static native String getBuildSHA1();
```

Descrição

Função utilizada para se obter a string do md5sum correspondente a build da biblioteca do VirtualLoop.

Parâmetros

Nenhum

Retorno

String do md5sum correspondente a build da biblioteca do VirtualLoop.

`loadLibrary`

Protótipo da Função

```
public static void loadLibrary() { System.loadLibrary("ptx_virtualLoopJava"); }
```

Descrição

Função que realiza a carga da biblioteca em Java.

Parâmetros

Nenhum

Retorno

Nenhum

4.3.2. API Cliente

```
//================================================================================
    // HELPER TYPES
    //================================================================================
    // JVirtualLoopHandleStatus
    public static class Status {
        public boolean isConnected;
    };
    // Event Callbacks
    public interface Callbacks {
        public void onEvent(VLCommon.Event event, byte[] jpeg);
        public void onError(int errorId);
        public void onFrame(byte[] jpeg);
    }
    // JVirtualLoopConfig
    public static class Config {
        public String mjpegUrl;
        public String serverIp;
        public int serverPort;
        public int frameQueueCap;
        public Callbacks callbacks;
    };
    //================================================================================
    // METHODS
    //================================================================================
    public VLClient() throws RuntimeException {
        this.nativeHandle = native_createHandle();
        if(this.nativeHandle == 0) {
            throw new RuntimeException();
        }
        System.out.println("Client created");
    }

    @Override
    public void close() throws Exception {
        native_destroyHandle(this.nativeHandle);
    }

    /* requests */
    public native int connect(Config config);
    public native int connectInfo(Config config, VLCommon.JVirtualLoopServerInfo info);
    public native int disconnect();
    public native int requestLastFrame();
    public native int setCalibRegion(VLCommon.CalibRegion cr);
    public native int addEntryRegion(VLCommon.EntryRegion er);
    public native int addActiveRegion(VLCommon.ActiveRegion ar);
    public native int addAbandonRegion(VLCommon.AbandonDetRegion abr);
    public native int setFloatProperty(int prop, float value);
    public native int getFloatProperty(int prop, VLCommon.MutableValue<Float> value);
    public native int enableDetection();

    /* status */
    public native Status getStatus();

    /* session file */
    public native int loadSessionFile(String filename);

    //================================================================================
    // PRIVATE NATIVE METHODS
    //================================================================================
    private native long native_createHandle();
    private native void native_destroyHandle(long handle);
    final private long nativeHandle;
```

4.3.2.1. Tipos

```
public static class Status
```

Descrição

Classe Java usada para retornar o status de conexão do cliente.

Membros

- `public boolean isConnected` : booleano representativo do estado de coneção.

```
public interface Callbacks
```

Descrição

Interface Java usada para descrever os diferentes tipo de callback da biblioteca.

Métodos

- `public void onEvent(VLCommon.Event event, byte[] jpeg)` : Callback de evento.
- `public void onError(int errorId)` : Callback de erro.
- `public void onFrame(byte[] jpeg)` : Callback de chegada de quadro.

`public static class Config`

Descrição

Classe Java usada para armazenar as configurações do cliente do VirtualLoop.

Membros

- `public String mjpegUrl` : string contendo a URL do fluxo Mjpeg no formato `http://<IP>[:PORT]/[PATH]`
- `public String serverIp` : endereço de IP do servidor no formato **A.B.C.D**
- `public int serverPort` : porta TCP utilizada pelo servidor
- `public int frameQueueCap` : tamanho máximo da fila de frames mantidos pelo servidor [valor recomendado: 100]
- `public Callbacks callbacks` : interface de contexto fornecida pelo usuário. Este ponteiro é passado como argumento nas chamadas das callbacks.

4.3.2.2. Métodos

`VLClient`

Protótipo da Função

```
    public VLClient();
```

Descrição

Função utilizada para criar um cliente VirtualLoop.

Parâmetros

Nenhum

Retorno

Nenhum

`close`

Protótipo da Função

```
@Override
    public void close()
```

Descrição

Função utilizada para destruir um `handle` cliente a um servidor de VirtualLoop.

Parâmetros

Nenhum

Retorno

Nenhum

`connect`

Protótipo da Função

```
public native int connect(Config config);
```

Descrição

Função utilizada para conectar um `handle` cliente a um servidor de VirtualLoop utilizando as configurações definidas em `config`.

Parâmetros

- `Config config` : Objeto de configuração válido

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

connectInfo

Protótipo da Função

```
public native int connectInfo(Config config, VLCommon.JVirtualLoopServerInfo info);
```

Descrição

Possui a mesma funcionalidade da função **connect** mas recebe um parâmetro adicional para receber informações sobre a licença e a versão do servidor.

Parâmetros

- `Config config` : Objeto de configuração válido
- `VLCommon.JVirtualLoopServerInfo info info` : Objeto que armazenará as informações de licença e a versão do servidor.

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

jl_vl_disconnect

Protótipo da Função

```
public native int disconnect();
```

Descrição

Função utilizada para desconectar um `handle` VirtualLoop. Esta função bloqueia até que a desconexão seja concluída e os eventos pendentes sejam executados.

Parâmetros

Nenhum

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

requestLastFrame

Protótipo da Função

```
public native int requestLastFrame();
```

Descrição

Função utilizada para solicitar o último quadro de vídeo processado pelo servidor. Esta função é assíncrona, retorna imediatamente e só tem efeito se o handle já estiver conectado ao servidor. A callback `on_frame_cb` é chamada caso o quadro seja recebido com sucesso. Caso o servidor não tenha recebido nenhum quadro até o momento da requisição, a callback `on_error_cb` é chamada com o código `PTX_LAST_FRAME_UNAVAILABLE`.

Devido à fila de frames do servidor, o frame retornado pode ser relativo a um momento do passado, não havendo garantias de sincronismo com o fluxo

Mjpeg. Além disso, chamadas consecutivas a esta função podem retornar um mesmo quadro caso nenhum novo quadro tenha sido recebido ou processado pelo servidor no intervalo entre as chamadas.

Parâmetros

Nenhum

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

setCalibRegion

Protótipo da Função

```
public native int setCalibRegion(VLCommon.CalibRegion cr);
```

Descrição

Função utilizada para definir a **CalibRegion** de uma instância do VirtualLoop. A definição desta região é obrigatória apenas nas aplicações que necessitem de medição de velocidade. Caso uma **CalibRegion** já tenha sido definida para um `handle`, está função retorna `PTX_API_CALL_HAS_NO_EFFECT`.

Ver **Demarcação da CalibRegion** para maiores informações.

Parâmetros

- `VLCommon.CalibRegion cr` : Objeto válido para uma **class CalibRegion** contendo as configurações da **CalibRegion**

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

addEntryRegion

Protótipo da Função

```
public native int addEntryRegion(VLCommon.EntryRegion er);
```

Descrição

Função utilizada para adicionar uma **EntryRegion** a uma instância do VirtualLoop. Um `handle` só pode adicionar uma **EntryRegion** após estar conectado a um servidor VirtualLoop, caso contrário a chamada retorna com erro `PTX_API_CALL_HAS_NO_EFFECT`. Para redefinir uma região, deve-se destruir e recriar o handle.

> **Atenção:** o número de **EntryRegions** influencia diretamente na carga de processamento do servidor VirtualLoop

Parâmetros

- `VLCommon.EntryRegion er` : Objeto válido para uma **class EntryRegion** contendo as configurações da **EntryRegion**

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

addActiveRegion

Protótipo da Função

```
public native int addActiveRegion(VLCommon.ActiveRegion ar);
```

Descrição

Função utilizada para adicionar uma **ActiveRegion** a uma instância do VirtualLoop. Um `handle` só pode adicionar uma **EntryRegion** após estar conectado a um servidor VirtualLoop, caso contrário a chamada retorna com erro `PTX_API_CALL_HAS_NO_EFFECT`. Para redefinir uma região, deve-se destruir e recriar o handle.

> **Atenção:** o número de **ActiveRegion** tem pouco impacto na carga de processamento do servidor VirtualLoop

Parâmetros

- `VLCommon.ActiveRegion ar` : Objeto válido para uma **class ActiveRegion** contendo as configurações da **ActiveRegion**

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

addAbandonRegion

Protótipo da Função

```
public native int addAbandonRegion(VLCommon.AbandonDetRegion abr);
```

Descrição

Função utilizada para adicionar uma **AbandonDetRegion** a uma instância do VirtualLoop. Um `handle` só pode adicionar uma **AbandonDetRegion** após estar conectado a um servidor VirtualLoop, caso contrário a chamada retorna com erro `PTX_API_CALL_HAS_NO_EFFECT`. Para redefinir uma região, deve-se destruir e recriar o handle.

> **Atenção:** o número de **AbandonDetRegion** impacto na carga de processamento do servidor VirtualLoop

Parâmetros

- `VLCommon.AbandonDetRegion ar` : Objeto válido para uma **struct JVirtualLoopAbandonDetRegion** contendo as configurações da **AbandonDetRegion**

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

setFloatProperty

Protótipo da Função

```
public native int setFloatProperty(int prop, float value);
```

Descrição

Altera o valor de uma variável do tipo float da configuração.

Parâmetros

- `int prop` : enum que identifica qual a propriedade a ser alterada. Ver **enum JVirtualLoopProperty**
- `float value` : valor que deverá ser atribuido à propriedade

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

getFloatProperty

Protótipo da Função

```
public native int getFloatProperty(int prop, VLCommon.MutableValue<Float> value);
```

Descrição

Lê o valor de uma variável do tipo float da configuração.

Parâmetros

- `int prop` : enum que identifica qual a propriedade a ser lida. Ver **enum JVirtualLoopProperty**

- `VLCommon.MutableValue<Float> value` : template para variável float onde será escrito o valor da propriedade

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

enableDetection

Protótipo da Função

```
public native int enableDetection();
```

Descrição

Esta função deve ser chamada após a configuração de todas as regiões de uma instância do VirtualLoop: `CalibRegion`, `EntryRegion`, `ActiveRegion` e `AbandonDetRegion`. Ela serve para indicar ao servidor que todas as configurações já foram feitas e a detecção pode começar. Nenhuma região pode ser adicionada após a chamada dessa função, sendo necessário destruir e recriar o handle caso seja necessário fazer alterações.

Veja o arquivo `VLSample.java` para um exemplo de uso desta função.

Parâmetros

Nenhum

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

getStatus

Protótipo da Função

```
public native Status getStatus();
```

Descrição

Função utilizada para recuperar o estado do handle cliente do VirtualLoop através de em uma **struct JVirtualLoopHandleStatus**.

Parâmetros

Nenhum

Retorno

- `PTX_SUCCESS` em caso de sucesso, ou um código de erro caso contrário - ver **Códigos de retorno de função**

---

loadSessionFile

Protótipo da Função

```
public native int loadSessionFile(String filename);
```

Descrição

Função utilizada para carregar as configurações dos laços à partir de um arquivo de sessão (*session file*) do VirtualLoop. Um *session file* é um arquivo JSON cujo formato é apresentado a seguir.

```
{
    "jidosha-light" : {
        "header" : {
            "er_cnt" : 1,
            "ar_cnt" : 2,
            "abr_cnt" : 0
        },
        "data" : {
            "cbw" : 1.0,
            "cbh" : 2.0,
            "cb"  : [x0,y0,x1,y1,x2,y2,x3,y3],
            "er0" : [x0,y0,x1,y1,x2,y2,x3,y3],
            "ar0" : [x0,y0,x1,y1,x2,y2,x3,y3],
```

```
                "ar1" : [x0,y0,x1,y1,x2,y2,x3,y3]
            },
            "help" : {
                "desc"  : "Este dicionário é fornecido apenas como especificação do formato do session file",
                "note"  : "Todos os pontos das reigões devem ser inseridos no formato [x0,y0,...,x3,y3]",
                "cb"    : "4 pontos da CalibRegion. Se este campo for omitido a CalibRegion é ignorada",
                "cbw"   : "Largura em metros da CalibRegion",
                "cbh"   : "Altura em metros da CalibRegion",
                "erN"   : "Pontos da n-ésima EntryRegion no intervalo [0..N-1]",
                "arN"   : "Pontos da n-ésima ActiveRegion no intervalo [0..N-1]",
                "abrN"  : "Pontos da n-ésima AbandonDetRegion no intervalo [0..N-1]",
                "er_cnt" : "Número de EntryRegions dentro de data. O parser procura pelo formato: er%d",
                "ar_cnt" : "Número de ActiveRegions dentro de data. O parser procura pelo formato: ar%d",
                "ar_cnt" : "Número de AbandonDetRegions dentro de data. O parser procura pelo formato: ar%d",
                "ar_df" : "ActiveRegion deb_frames",
                "ar_sf" : "ActiveRegion stop_frames",
                "ar_st" : "ActiveRegion sense_thresh"
            }
        }

    }
```

Parâmetros

- `String filename filename` : caminho para o arquivo de configuração

Retorno

- `PTX_SUCCESS` caso as configurações da sessão sejam carregadas com sucesso ou um código de erro - ver **Códigos de retorno de função**

---

native_createHandle

Protótipo da Função

```
    private native long native_createHandle();
```

Descrição

Função utilizada para criar um handle cliente VirtualLoop. Utilizada em **VLClient()**

Parâmetros

Nenhum

Retorno

- `nativeHandle`

---

native_destroyHandle

Protótipo da Função

```
    private native void native_destroyHandle(long handle);
```

Descrição

Função utilizada para destruir um handle cliente VirtualLoop. Utilizada em **close()**

Parâmetros

- `long handle` : handle cliente VirtualLoop

Retorno

Nenhum

4.3.3. API Servidor

```
    //=========================================================================
    // METHODS
    //=========================================================================
    public VLServer(int port, int conns) throws Exception {
        if(port < 0 || conns < 1) {
            throw new IllegalArgumentException("port < 0 || conns < 1");
        }
        this.port = port;
        this.conns = conns;
        this.nativeHandle = native_createHandle(port,conns);
        if(this.nativeHandle == 0) {
            throw new RuntimeException();
        }
        System.out.println("Server Created");
    }

    @Override
    public void close() throws Exception {
        native_destroyHandle(this.nativeHandle);
    }

    //=========================================================================
    // PRIVATE NATIVE METHODS
    //=========================================================================
    private native long native_createHandle(int port, int conns);
    private native void native_destroyHandle(long handle);
    final private long nativeHandle;
```

4.3.3.1. Métodos

`VLServer`

Protótipo da Função

```
    public VLServer(int port, int conns);
```

Descrição

Função utilizada para criar um e inicializar servidor VirtualLoop.

Parâmetros

- `int port`: valor da porta TCP utilizada pelo servidor
- `int conns`: número máximo de conexões simultâneas que o servidor pode aceitar (o valor máximo é limitado pela licença)

Retorno

Nenhum

`close`

Protótipo da Função

```
@Override
    public void close()
```

Descrição

Função utilizada para interromper e destruir um servidor VirtualLoop.

Parâmetros

Nenhum

Retorno

Nenhum

`native_createHandle`

Protótipo da Função

```
    private native long native_createHandle(int port, int conns);
```

Descrição

Função utilizada para criar um handle servidor VirtualLoop. Utilizada em **VLServer()**

Parâmetros

- `int port`: valor da porta TCP utilizada pelo servidor
- `int conns`: número máximo de conexões simultâneas que o servidor pode aceitar (o valor máximo é limitado pela licença)

Retorno

- `nativeHandle`

---

native_destroyHandle

Protótipo da Função

```
    private native void native_destroyHandle(long handle);
```

Descrição

Função utilizada para destruir um handle cliente VirtualLoop. Utilizada em **close()**

Parâmetros

- `long handle` : handle cliente VirtualLoop

Retorno

Nenhum

4.3.4. Códigos de retorno de função

Os código de retorno de função são compartilhados entre as bibliotecas desenvolvidas pela **Pumatronix Equipamentos Eletrônicos** e estão definidos no header `ptx/common/ptx_codes.h`. Os códigos relevantes para o **VirtualLoop** são os seguintes:

- `PTX_SUCCESS` : Ação retornou com sucesso
- `PTX_INVALID_IMAGE_TYPE` : Tipo inválido de imagem fornecido
- `PTX_INVALID_PARAMETER` : Parâmetro inválido fornecido
- `PTX_API_CALL_NOT_SUPPORTED` : Chamada não suportada pela biblioteca para a plataforma
- `PTX_INVALID_ROI` : Os pontos fornecidos para a ROI são inválidos
- `PTX_INVALID_HANDLE` : O ponteiro para o handle fornecido é inválido
- `PTX_API_CALL_HAS_NO_EFFECT` : A chamada não teve efeito `no-op` - algumas chamadas possuem precedência, veja a documentação do método
- `PTX_LICENSE_INVALID` : A fonte de licença é inválida
- `PTX_LICENSE_EXPIRED` : A fonte de licença expirou
- `PTX_LICENSE_MAX_THREADS_EXCEEDED` : O número máximo de threads de processamento foi excedido
- `PTX_LICENSE_UNTRUSTED_RTC` : A fonte de tempo do sistema não é confiável (licença RTC)
- `PTX_LICENSE_MAX_CONNS_EXCEEDED` : O número máximo de conexões de processamento foi excedido
- `PTX_LICENSE_UNAUTHORIZED_PRODUCT` : O produto não está autorizado pela licença
- `PTX_CONNECT_FAILED` : Não foi possível concluir a conexão com o servidor
- `PTX_SOCKET_DISCONNECT` : A conexão com o servidor foi interrompida de maneira forçada
- `PTX_SOCKET_QUEUE_TIMEOUT` : Timeout não fila de requisições da conexão
- `PTX_SOCKET_QUEUE_FULL` : Fila de requisições cheia
- `PTX_SOCKET_IO_ERROR` : Erro de escrita ou leitura do socket
- `PTX_SOCKET_WRITE_FAILED` : Erro de escrita do socket
- `PTX_SOCKET_READ_TIMEOUT` : Tempo máximo para a recepção excedido
- `PTX_INVALID_RESPONSE` : Resposta recebida veio falformada
- `PTX_HANDLE_QUEUE_FULL` : File de requisições do handle está cheia
- `PTX_INVALID_REQUEST` : Requisição inválida
- `PTX_INVALID_MESSAGE` : Mensagem de requisição inválida
- `PTX_INVALID_STREAM_FRAME` : Frame inválido
- `PTX_FRAME_QUEUE_FULL` : File de frames cheia
- `PTX_LAST_FRAME_UNAVAILABLE` : Frame indisponível no momento
- `PTX_SERVER_CONN_LIMIT_REACHED` : Número máximo de conexões ao servidor excedido
- `PTX_MJPEG_ERROR_BASE` : Offset para os erros relativos ao Mjpeg (este valor nunca é utilizado como erro)
- `PTX_MJPEG_HTTP_HEADER_OVERFLOW` : O cabeçalho da resposta à requisição HTTP do Mjpeg veio corrompido
- `PTX_MJPEG_HTTP_RESPONSE_NOT_OK` : O servidor Mjpeg não aceitou a requisição HTTP
- `PTX_MJPEG_HTTP_CONTENT_TYPE_ERROR` : O servidor Mjpeg respondeu a requisição com `content-type` inválido
- `PTX_MJPEG_HTTP_CONTENT_LENGTH_ERROR` : O servidor Mjpeg respondeu a requisição com `content-length` inválido
- `PTX_MJPEG_HTTP_FRAME_BOUNDARY_NOT_FOUND` : O servidor Mjpeg não adicionou corretamente o `frame-boundary` ao quadro
- `PTX_MJPEG_CONNECTION_CLOSED` : A conexão foi encerrada pelo servidor Mjpeg
- `PTX_MJPEG_CONNECT_FAILED` : Não foi possível conectar ao servidor Mjpeg
- `PTX_UNKNOWN_ERROR` : Erro desconhecido, contate **contato@pumatronix.com.br**

4.4. API VirtualLoop CSharp

A API C# está contida no arquivo **VLApi.cs** e um exemplo de seu uso pode ser encontrado no arquivo **VLSample.cs**.

A implementação da API é dividida em duas camadas: **Native** e **High-Level**, sendo a primeira utiliza apenas internamento pelo wrapper para adaptar o acesso à API C. É aconselhável que apenas a API **High-Level** seja utilizada pelo usuário.

Os métodos públicos e as contantes relevante da API **High-Level** estão descritos à seguir. A interpretação dos códigos de retorno, constantes e métodos segue a mesma documentação da API C.

4.4.1. API Alto-Nível (High Level)

```
namespace VL {
    namespace Common {
    public class ReturnCode {
        // Available codes
        public const int SUCCESS                            = 0;
        public const int FILE_NOT_FOUND                     = 1;
        public const int INVALID_IMAGE                      = 2;
        public const int INVALID_IMAGE_TYPE                 = 3;
        public const int INVALID_PARAMETER                  = 4;
        public const int COUNTRY_NOT_SUPPORTED              = 5;
        public const int API_CALL_NOT_SUPPORTED             = 6;
```

```
        public const int INVALID_ROI                          = 7;
        public const int INVALID_HANDLE                       = 8;
        public const int API_CALL_HAS_NO_EFFECT               = 9;
        public const int INVALID_IMAGE_SIZE                   = 10;
        public const int LICENSE_INVALID                      = 16;
        public const int LICENSE_EXPIRED                      = 17;
        public const int LICENSE_MAX_THREADS_EXCEEDED         = 18;
        public const int LICENSE_UNTRUSTED_RTC                = 19;
        public const int LICENSE_MAX_CONNS_EXCEEDED           = 20;
        public const int LICENSE_UNAUTHORIZED_PRODUCT         = 21;
        public const int CONNECT_FAILED                       = 100;
        public const int SOCKET_DISCONNECT                    = 101;
        public const int SOCKET_QUEUE_TIMEOUT                 = 202;
        public const int SOCKET_QUEUE_FULL                    = 103;
        public const int SOCKET_IO_ERROR                      = 104;
        public const int SOCKET_WRITE_FAILED                  = 105;
        public const int SOCKET_READ_TIMEOUT                  = 106;
        public const int INVALID_RESPONSE                     = 107;
        public const int HANDLE_QUEUE_FULL                    = 108;
        public const int INVALID_REQUEST                      = 109;
        public const int INVALID_MESSAGE                      = 110;
        public const int INVALID_STREAM_FRAME                 = 209;
        public const int FRAME_QUEUE_FULL                     = 211;
        public const int LAST_FRAME_UNAVAILABLE               = 212;
        public const int SERVER_CONN_LIMIT_REACHED            = 213;
        public const int SERVER_VERSION_NOT_SUPPORTED         = 214;
        public const int MJPEG_ERROR_BASE                     = 1000;
        public const int MJPEG_HTTP_HEADER_OVERFLOW           = 1001;
        public const int MJPEG_HTTP_RESPONSE_NOT_OK           = 1002;
        public const int MJPEG_HTTP_CONTENT_TYPE_ERROR        = 1003;
        public const int MJPEG_HTTP_CONTENT_LENGTH_ERROR      = 1004;
        public const int MJPEG_HTTP_FRAME_BOUNDARY_NOT_FOUND  = 1005;
        public const int MJPEG_CONNECTION_CLOSED              = 1006;
        public const int MJPEG_CONNECT_FAILED                 = 1007;
        public const int HW_FPGA_INIT_FAILED                  = 2000;
        public const int HW_FPGA_LOCK_FAILED                  = 2001;
        public const int UNKNOWN_ERROR                        = 99999;
        // Helper ToString method
        public static string ToString(int code);
    }
    };

    //================================================================================
    // Enums
    //================================================================================
    public class RegionState {
        public const int PTX_VL_REGION_STATE_IDLE                 = 0;
        public const int PTX_VL_REGION_STATE_ACTIVE               = 1;
        public const int PTX_VL_REGION_STATE_HOLDING              = 2;
        public const int PTX_VL_REGION_STATE_LINE_STOP            = 3;
        public const int PTX_VL_REGION_INVALID                    = 4;
    };

    public class EventExtraField {
        public const int PTX_VL_EVENT_SPEED                       = 0;
        public const int PTX_VL_EVENT_COUNTER_TOTAL               = 1;
        public const int PTX_VL_EVENT_COUNTER_SAME_ORIGIN         = 2;
        public const int PTX_VL_EVENT_VEHICLE_TYPE                = 3;
        public const int PTX_VL_EVENT_COUNTER_DELTA               = 4;
    };

    public class VehicleTypes {
        public const int PTX_VL_VEHICLE_TYPE_UNKNOWN              = 0;
        public const int PTX_VL_VEHICLE_TYPE_CAR                  = 1;
        public const int PTX_VL_VEHICLE_TYPE_MOTORCYCLE           = 2;
        public const int PTX_VL_VEHICLE_TYPE_TRUCK                = 3;
        public const int PTX_VL_VEHICLE_TYPE_BUS                  = 4;
    };

    public class VirtualLoopProperty {
        public const int PTX_VL_PROPERTY_VEHICLE_TYPE_ENABLED     = 0;
        public const int PTX_VL_PROPERTY_AUTHORIZED_VEHICLE_TYPES = 1;
        public const int PTX_VL_PROPERTY_ACTIVE_REGION_TIMEOUT    = 2;
    };


    //================================================================================
    // HELPER TYPES
    //================================================================================
    public class Point2i {
        public int mX;
        public int mY;
        public Point2i(int x,int y);
        public override string ToString();
    };

    public class Timestamp {
        public ushort mYear;
        public ushort mMonth;
        public ushort mDay;
        public ushort mHour;
        public ushort mMin;
        public ushort mSec;
        public ushort mMsec;
        public Timestamp([In] Native.Timestamp p);
    };

    public class Version {
        public int mMajor;
        public int mMinor;
        public int mRelease;

        public Version(int a, int b, int c);

        public override string ToString();
    };
```

```
    //============================================================================
    // MAIN TYPES
    //============================================================================
    public class EntryRegion {
        public uint mId;
        public List<Point2i> mPoints;

        public EntryRegion([In] Native.EntryRegion p);
        public EntryRegion(uint id, List<Point2i> points);

        public override string ToString();
    };

    public class ActiveRegion {
        public uint mDebFrames;
        public uint mStopFrames;
        public uint mSenseThresh;
        public uint mId;
        public List<Point2i> mPoints;

        public ActiveRegion([In] Native.ActiveRegion p);

        public ActiveRegion(uint id, List<Point2i> points, uint debFrames,
                            uint stopFrames, uint senseThresh);

        public override string ToString();
    };

    public class AbandonDetRegion {
        public uint mId;
        public List<Point2i> mPoints;

        public AbandonDetRegion([In] Native.AbandonDetRegion p);
        public AbandonDetRegion(uint id, List<Point2i> points);

        public override string ToString();
    };

    public class CalibRegion {
        public float mWidthInMeters;  /* width in meters of the rectangle */
        public float mHeightInMeters; /* height in meters of the rectangle */
        public List<Point2i> mPoints;

        public CalibRegion(List<Point2i> points, float widthInMeters, float heightInMeters);
        public CalibRegion([In] Native.CalibRegion p);

        public override string ToString();
    };

    public class AbandonObject {
        public uint mId;
        public uint mX;
        public uint mY;
        public uint mWidth;
        public uint mHeight;
        public AbandonObject([In] Native.AbandonObject p);
    };

    public class Event {
        public EntryRegion mOrigin;
        public ActiveRegion mTarget;
        public Timestamp mTs;
        public int mState;
        public Dictionary<int, float> mExtras;
        public Event([In] Native.Event p);
    };

    //============================================================================
    // High-level API [Common]
    //============================================================================
    public class Library {

        // Static information methods
        public static Common.Version getVersion();
        public static string getSHA1();
    };

    public class VirtualLoopLicenseInfo {
        public ulong mSerial;
        public string mCustomer;
        public int mMaxThreads;
        public int mMaxConnections;
        public int mState;
        public int mTtl;

        public VirtualLoopLicenseInfo([In] Native.VirtualLoopLicenseInfo p);
    };

    public class VirtualLoopServerInfo {
        public VirtualLoopLicenseInfo mLicense;
        public Version mVersion;
        public string mSha1;

        public VirtualLoopServerInfo();
        public VirtualLoopServerInfo([In] Native.VirtualLoopServerInfo p);
    };

    } /* end namespace Common */

    //============================================================================
    // High-level API [Server]
    //============================================================================
    namespace Server
    {
    public class VLServer : Common.NativeHandle {

        //============================================================================
```

```
        // ATTRIBUTES
        //========================================================================
        public int mPort;
        public int mConns;

        //========================================================================
        // METHODS
        //========================================================================
        public VLServer(int port, int conns);

    };
    } /* end namespace Server */

    namespace Client
    {

    //========================================================================
    // TYPES
    //========================================================================
    // Config
    public class Config {
        public string mMjpegUrl;
        public string mServerIp;
        public int    mServerPort;
        public int    mFrameQueueCap;
        public ICallbacks mCallback;
    };

    public class Status {
        public bool isConnected;
    };

    // Event Callbacks
    public interface ICallbacks {
        /* Recognition result callback function pointer */
        void OnEvent(Common.Event evt, byte[] jpeg);
        void OnAbandon(Common.AbandonObject regions, uint regions_size, byte[] jpeg);
        void OnError(uint errorId);
        void OnFrame(byte[] jpeg);
    }

    public class VLClient : VL.Common.NativeHandle {

        //========================================================================
        // METHODS
        //========================================================================
        public VLClient();

        /* requests */
        public int connect(Client.Config config);
        public int connectInfo(Client.Config config, ref Common.VirtualLoopServerInfo info);
        public int disconnect();

        public int requestLastFrame();
        public int setCalibRegion(Common.CalibRegion cr);
        public int addEntryRegion(Common.EntryRegion er);
        public int addActiveRegion(Common.ActiveRegion ar);
        public int addAbandonRegion(Common.AbandonDetRegion abr);
        public int enableDetection();

        /* status */
        public Status getStatus();

        /* session file */
        public int loadSessionFile(string filename);

        /* setters and getters */
        public int setFloatProperty(int prop, float value);
        public int getFloatProperty(int prop, ref float value);

    };
    } /* end namespace Client */
    } /* end namespace VL */
```

Índice da API

C/C++

JAVA

DELPHI