Jidosha Container

Manual do Usuário

Biblioteca de software para reconhecimento de códigos de container

Release: v2.0.1
Data: 01/04/2021

Table of Contents

Releases

Date	Version	Revision
29/12/2020	2.0.0	• Reestruturação interna da biblioteca • Melhorias na estabilidade da detecção do hardkey • Suporte à arquitetura ARMv8 (AARCH64)

1. Visão Geral

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 Ltda. 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

Este presente documento **Manual do Usuário – JIDOSHA CONTAINER** tem por objetivo detalhar as funções da biblioteca de software especializada em reconhecimento de códigos de container padrão ISO6436, chamada JIDOSHA CONTAINER, e suas condições de uso para correto funcionamento.

2.1. Objetivo

A biblioteca de software JIDOSHA CONTAINER tem como principal funcionalidade reconhecer códigos de containers no padrão ISO6436 a partir de imagens. Sua principal aplicação é na gestão e controle de acesso aos portos, cenário para qual foi criada e no qual apresenta um excelente desempenho. Porém, é possível usar a biblioteca em qualquer tipo de aplicação onde seja necessária a leitura deste tipo de padrão, por exemplo em pátios de armazenagem de containers.

Com um alto índice de reconhecimento, o JIDOSHA CONTAINER é a ferramenta ideal para quem necessita ter informação dos códigos de container de forma automática, sem intervenções externas, através de métodos de análise de imagem.

2.2. Condições de Uso

A biblioteca de software JIDOSHA CONTAINER foi criada para funcionar em conjunto com o *hardkey* (chave de segurança) que acompanha a biblioteca. Ou seja, para correto funcionamento da biblioteca o referido *hardkey* deverá estar conectado à USB do ambiente onde a biblioteca estará sendo utilizada.

Há duas versões de *hardkey*, uma de demonstração e outra liberada. A versão demonstração tem data de validade, enquanto a versão liberada não. Quando a data de validade expira, a bilbioteca automaticamente passa a retornar códigos vazios. Se seu *hardkey* de demonstração expirou e você deseja comprar uma licença ou estender o período de demonstração, entre em contato com a Pumatronix Equipamentos Eletrônicos (**contato@pumatronix.com.br**).

Esta versão do JIDOSHA CONTAINER possui compatibilidade com sistema operacional Windows com plataforma Delphi, C++ Builder 6, Visual C++, .NET 2.0 ou superior, e Java; e com sistema operacional Linux com plataforma C (gcc), C++ (g++) e Java.

3. Instalação

3.1. Windows

Para instalar a biblioteca de software JIDOSHA CONTAINER, basta descompactar o arquivo de SDK (Software Development Kit) fornecido, que deverá conter DLLs e código fonte de demonstração, e copiar as DLLs para a pasta do seu projeto. Caso alguma senha seja exigida ao tentar descompactar o arquivo, a senha padrão é “jidosha”.

Para testar o funcionamento da biblioteca e do *hardkey*, plugue o *hardkey* (o Windows irá instalar um driver automaticamente na primeira vez), abra uma janela do prompt de comando (cmd.exe) e, de dentro da pasta onde descompactou o SDK, digite “containerSample.exe 0 0 Container_teste.jpg”.

Se tudo correr bem, esse programa deve retornar, entre outras informações, a palavra “autorizado” e o código contido na imagem. Caso a palavra “autorizado” não apareça, experimente remover o *hardkey* e plugá-lo novamente. Caso isso não resolva o problema, sua versão do *hardkey* pode ter expirado, ou pode haver algum problema com o mesmo. Nesse caso, entre em contato com a Pumatronix Equipamentos Eletrônicos.

3.2. Linux

3.2.1. Configuração das permissões do *hardkey*

Para o correto funcionamento do *hardkey* USB, 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
```

Já para 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 a Pumatronix Equipamentos Eletrônicos.

4. API

4.1. API JIDOSHA CONTAINER (C/C++)

A API (Application Programming Interface) nativa da biblioteca está escrita em linguagem C, o que permite seu uso a partir de qualquer linguagem. O SDK inclui bibliotecas wrapper para simplificar o uso da biblioteca a partir de .NET (C# e VB.NET), Java e Delphi. Esses wrappers simplesmente encapsulam as chamadas às funções da biblioteca, fazendo qualquer conversão necessária de parâmetros e resultados.

Toda a API C está disponível através de um único arquivo header, containerCore.h, cujo conteúdo comentado é apresentado a seguir. Uma descrição mais detalhada também é apresentada.

A biblioteca pode ser usada de duas formas: através da API 1 ou da API 2. A API 1, que foi a primeira API do JIDOSHA CONTAINER, tem como principal motivação a facilidade de uso. É possível ler códigos através de uma única chamada de função (`containerReadText` ou `containerReadTextFromMemory`, no caso de linguagem C).

Já a API 2 foi criada para proporcionar maior flexibilidade na configuração da biblioteca e na carga de imagens. Por exemplo, é possível configurar o número mínimo de caracteres que devem ser lidos com confiabilidade boa para o código ser considerado válido. É possível adicionar novos parâmetros de configuração à API 2 sem afetar usuários existentes da biblioteca (ou seja, estes usuários podem atualizar a DLL/.so do JIDOSHA CONTAINER para uma versão mais recente, sem precisar recompilar).

Recomendamos a API 1 para quem precisa integrar o JIDOSHA CONTAINER à sua aplicação o mais rapidamente possível, e a API 2 para quem gostaria de maior controle sobre o funcionamento da biblioteca.

containerCore.h

```
#define MAX_NUM_CHARS 11

    #define ContainerHandle void
    #define ContainerImage void

    enum containerFlip {
        CONTAINER_FLIP_0 = 0,
        CONTAINER_FLIP_180 = 1,
        CONTAINER_FLIP_AMBOS = 2,
    };

    enum containerError {
        CONTAINER_SUCCESS = 0,
        CONTAINER_ERROR_HARDKEY_NOT_FOUND = 1,
        CONTAINER_ERROR_HARDKEY_NOT_AUTHORIZED = 2,
        CONTAINER_ERROR_FILE_NOT_FOUND = 3,
        CONTAINER_ERROR_INVALID_IMAGE = 4,
        CONTAINER_ERROR_INVALID_IMAGE_TYPE = 5,
        CONTAINER_ERROR_INVALID_PROPERTY = 6,
        CONTAINER_ERROR_COUNTRY_NOT_SUPPORTED = 7,
        CONTAINER_ERROR_OTHER = 999,
    };

    enum containerImageType {
        CONTAINER_IMAGE_TYPE_STRUCTURED = 0,
        CONTAINER_IMAGE_TYPE_RAW_GRAY8 = 1,
        CONTAINER_IMAGE_TYPE_RAW_RGB = 2,
        CONTAINER_IMAGE_TYPE_RAW_BGR = 3,
    };

    typedef struct ContainerConfig
    {
        int timeout;    // timeout em milisegundos
        int flip;       // 0 - 0 graus, 1 - 180 graus, 2 - as duas
    } ContainerConfig;

    typedef struct ContainerRecognition
    {
        char text[MAX_NUM_CHARS+1];
        double probabilities[MAX_NUM_CHARS];
        int xText;
        int yText;
        int widthText;
        int heightText;
        int xChar[MAX_NUM_CHARS];
        int yChar[MAX_NUM_CHARS];
        int widthChar[MAX_NUM_CHARS];
        int heightChar[MAX_NUM_CHARS];
        int textColor;
    } ContainerRecognition;

    typedef struct ContainerResultList
    {
        struct ContainerResultList* next;
        struct ContainerRecognition* recognition;
    } ContainerResultList;

    JIDOSHACORE_API int containerReadTextFromMemory(const unsigned char* stream, int n, ContainerConfig* config, ContainerRecogr
    JIDOSHACORE_API int containerReadText(const char* filename, ContainerConfig* config, ContainerRecognition* rec);
    JIDOSHACORE_API int containerGetVersion(int* major, int* minor, int* release);
    JIDOSHACORE_API int containerGetHardkeySerial(unsigned long* serial);
    JIDOSHACORE_API int containerGetHardkeyState(int* state);

    JIDOSHACORE_API int containerGetHardkeyRemainingTime(int* days, int* hours);

    JIDOSHACORE_API const char* containerBuildInfo();
    JIDOSHACORE_API int containerNumThreads();
```

```
// API 2
JIDOSHACORE_API ContainerHandle* containerInit();
JIDOSHACORE_API int containerDestroy(ContainerHandle* handle);
JIDOSHACORE_API int containerLoadImage(const char* filename, ContainerImage** img);
JIDOSHACORE_API int containerLoadImageFromMemory(unsigned char* stream, int length, int type, int width, int height, Contair
JIDOSHACORE_API int containerFreeImage(ContainerImage** img);
JIDOSHACORE_API int containerSetIntProperty(ContainerHandle* handle, const char* name, int value);
JIDOSHACORE_API int containerGetIntProperty(ContainerHandle* handle, const char* name, int* value);
JIDOSHACORE_API int containerSetDoubleProperty(ContainerHandle* handle, const char* name, double value);
JIDOSHACORE_API int containerGetDoubleProperty(ContainerHandle* handle, const char* name, double* value);
JIDOSHACORE_API int containerSetCharProperty(ContainerHandle* handle, const char* name, char value);
JIDOSHACORE_API int containerGetCharProperty(ContainerHandle* handle, const char* name, char* value);
JIDOSHACORE_API int containerFreeResultList(ContainerResultList* list);
JIDOSHACORE_API int containerFindFirst(ContainerHandle* handle, ContainerImage* img, ContainerResultList* list);
JIDOSHACORE_API int containerFindNext(ContainerHandle* handle, ContainerImage* img, ContainerResultList* list);
```

4.1.1. API 1

4.1.1.1 Structs

`struct ContainerConfig`

Descrição

A finalidade dessa estrutura é configurar o comportamento da biblioteca na chamada de reconhecimento de código.

Membros

`int timeout`: indica o tempo máximo que o reconhecimento deve levar, em milisegundos. Um valor de zero indica que não há timeout. Um valor diferente de zero ajuda a manter baixo o tempo médio de processamento. O valor deve ser determinado com base na resolução da imagem e CPU utilizada.

`int flip`: indica a orientação em que o código pode ser encontrado na imagem, devendo ser um dentre os seguintes valores:

CONTAINER_FLIP_0: O código está na orientação natural da imagem.

CONTAINER_FLIP_180: O código está invertido com relação à imagem. Recomendado para casos onde a câmera está instalada rotacionada em 180 graus.

CONTAINER_FLIP_AMBOS: O código pode estar normal ou invertido. Recomendado para o caso de leitura de códigos no topo de containers, onde ocorrem ambas as orientações.

`struct ContainerRecognition`

Descrição

A finalidade dessa estrutura é guardar o resultado do reconhecimento do código, incluindo: os caracteres do código, a confiabilidade de cada caractere, e as coordenadas do código na imagem.

Membros

`char text[MAX_NUM_CHARS+1]`: Código de MAX_NUM_CHARS caracteres terminada com '\0', ou string vazia se o código não foi encontrado.

`double probabilities[MAX_NUM_CHARS]`: valores de 0.0 a 1.0 indicando a confiabilidade, na forma de probabilidade, do reconhecimento de cada caractere.

`int xText` e `int yText`: coordenadas do ponto da esquerda superior do código, caso tenha sido encontrado.

`int widthText`: largura do retângulo que contém o código.

`int heightText`: altura do retângulo que contém o código.

`int xChar[MAX_NUM_CHARS]` e `int yChar[MAX_NUM_CHARS]`: coordenadas do ponto da esquerda superior de cada caractere do código, caso tenha sido encontrado.

`int widthChar[MAX_NUM_CHARS]`: largura do retângulo de cada caractere.

`int heightChar[MAX_NUM_CHARS]`: altura do retângulo de cada caractere.

`int textColor`: cor do texto do código, 0 - escuro, 1 - claro.

4.1.1.2 Funções

`containerReadText`

Protótipo da Função

```
int containerReadText(const char* filename, ContainerConfig* config, ContainerRecognition* rec);
```

Descrição

Reconhece o código na imagem e o guarda em um objeto `ContainerRecognition`. A imagem deverá ser passada como parâmetro no formato de path de onde está localizada a imagem. Caso não seja encontrado nenhum código, ou caso o hardkey não esteja autorizado ou não foi encontrado, o objeto `ContainerRecognition` conterá uma string vazia como código.

O arquivo de imagem deverá ser um bitmap, jpeg ou png.

Parâmetros

`filename`: path para o arquivo da imagem.

`config`: ponteiro para a struct `ContainerConfig` com a configuração para a biblioteca.

`rec`: ponteiro para a struct `ContainerRecognition` onde será armazenado o resultado da leitura.

Retorno

Código de erro: CONTAINER_SUCCESS no caso de sucesso, algum valor de containerError caso contrário.

---

containerReadTextFromMemory

Protótipo da Função

```
int containerReadTextFromMemory(const unsigned char* stream, int n, ContainerConfig* config, ContainerRecognition* rec);
```

Descrição

Reconhece o código na imagem e o guarda em um objeto `ContainerRecognition`. A imagem deverá ser passada como parâmetro no formato de array de bytes, e o número de bytes indicado pelo parâmetro `n`. Caso não seja encontrado nenhum código, ou caso o hardkey não esteja autorizado ou não foi encontrado, o objeto `ContainerRecognition` conterá uma string vazia como código.

O arquivo de imagem deverá ser um bitmap, jpeg ou png.

Parâmetros

`stream`: array de bytes que contém a imagem.

`n`: tamanho do array de bytes.

`config`: ponteiro para a struct `ContainerConfig` com a configuração para a biblioteca.

`rec`: ponteiro para a struct `ContainerRecognition` onde será armazenado o resultado da leitura.

Retorno

Código de erro: CONTAINER_SUCCESS no caso de sucesso, algum valor de containerError caso contrário.

---

containerGetVersion

Protótipo da Função

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

Descrição

Usada para verificar a versão da biblioteca, no formato major.minor.release.

Parâmetros

`major`, `minor` e `release`: ponteiros para variáveis int onde serão escritos os números que compõem a versão.

Retorno

Sempre retorna CONTAINER_SUCCESS.

`containerGetHardkeySerial`

Protótipo da Função

```
int getHardkeySerial(unsigned long* serial);
```

Descrição

Usada para verificar o número serial do hardkey.

Parâmetros

`serial`: ponteiro para variável unsigned long onde o número de série do hardkey será escrito.

Retorno

CONTAINER_SUCCESS em caso de sucesso, CONTAINER_ERROR_HARDKEY_NOT_FOUND caso o hardkey não tenha sido encontrado.

`containerGetHardkeyState`

Protótipo da Função

```
int getHardkeyState(int* state);
```

Descrição

Usada para verificar o estado do hardkey. Se state é igual a 0, o hardkey não está autorizado; se state é igual a 1, o hardkey está autorizado.

Parâmetros

`state`: ponteiro para variável int o estado do hardkey será escrito.

Retorno

CONTAINER_SUCCESS em caso de sucesso, CONTAINER_ERROR_HARDKEY_NOT_FOUND caso o hardkey não tenha sido encontrado.

`containerGetHardkeyRemainingTime`

Protótipo da Função

```
int getHardkeyRemainingTime(int* days, int* hours);
```

Descrição

Usada para verificar o tempo restante para licenças de demonstração. Se days e hours são iguais a -1 não há limite de tempo.

Parâmetros

`days`: ponteiro para variável int onde será escrito o número de dias restantes.

`hours`: ponteiro para variável int onde será escrito o número de horas restantes.

Retorno

CONTAINER_SUCCESS em caso de sucesso, CONTAINER_ERROR_HARDKEY_NOT_FOUND caso o hardkey não tenha sido encontrado.

4.1.2. API 2

4.1.2.1 Structs

`struct ContainerResultList`

Descrição

A finalidade dessa estrutura é armazenar a lista encadeada com os resultados dos processamentos das funções `containerFindFirst` e `containerFindNext`.

Membros

`struct ContainerResultList* next`: ponteiro para o próximo nó na lista. NULL se o nó atual é o último.

`struct ContainerRecognition* recognition`: ponteiro para o struct que contém um resultado de reconhecimento de código.

`typedef void ContainerHandle`

Descrição

Tipo utilizado para representar a memória alocada para a configuração.

`typedef void ContainerImage`

Descrição

Tipo utilizado para representar a memória alocada para uma imagem.

4.1.2.2 Funções

`containerFreeResultList`

Protótipo da Função

```
    void containerFreeResultList(ContainerResultList* list);
```

Descrição

Libera a memória alocada para lista encadeada de resultados.

Parâmetros

`list`: ponteiro para um struct `ContainerResultList`.

Retorno

Não possui retorno.

`containerInit`

Protótipo da Função

```
    ContainerHandle* containerInit();
```

Descrição

Aloca memória para a configuração da biblioteca. No caso de uso multithread, cada thread deverá chamar `containerInit` e usar seu próprio `ContainerHandle`.

Parâmetros

Não possui.

Retorno

Retorna um ponteiro para um `ContainerHandle` que será utilizado nas chamadas das funções subsequentes.

`containerDestroy`

Protótipo da Função

```
    int containerDestroy(ContainerHandle* handle);
```

Descrição

Libera a memória alocada pela função containerInit.

Parâmetros

`handle`: ponteiro para uma variável `ContainerHandle`.

Retorno

Sempre retorna CONTAINER_SUCCESS.

---

containerSetIntProperty

Protótipo da Função

```
int containerSetIntProperty(ContainerHandle* handle, const char* name, int value);
```

Descrição

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

Parâmetros

`handle`: ponteiro para um `ContainerHandle`.

`name`: string que contém o nome da propriedade a ser alterada.

`value`: valor que deverá ser atribuido à propriedade.

Retorno

CONTAINER_SUCCESS caso o valor da variável seja alterado, CONTAINER_ERROR_INVALID_PROPERTY caso a propriedade não exista ou não seja do tipo int.

---

containerGetIntProperty

Protótipo da Função

```
int containerGetIntProperty(ContainerHandle* handle, const char* name, int* value);
```

Descrição

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

Parâmetros

`handle`: ponteiro para um `ContainerHandle`.

`name`: string que contém o nome da propriedade a ser lida.

`value`: ponteiro para variável int onde será escrito o valor da propriedade.

Retorno

CONTAINER_SUCCESS caso o valor da variável seja lido, CONTAINER_ERROR_INVALID_PROPERTY caso a propriedade não exista ou não seja do tipo int.

---

containerSetDoubleProperty

Protótipo da Função

```
int containerSetDoubleProperty(ContainerHandle* handle, const char* name, double value);
```

Descrição

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

Parâmetros

`handle`: ponteiro para um `ContainerHandle`.

`name`: string que contém o nome da propriedade a ser alterada.

`value`: valor que deverá ser atribuido à propriedade.

Retorno

CONTAINER_SUCCESS caso o valor da variável seja alterado, CONTAINER_ERROR_INVALID_PROPERTY caso a propriedade não exista ou não seja do tipo double.

---

containerGetDoubleProperty

Protótipo da Função

```
int containerGetDoubleProperty(ContainerHandle* handle, const char* name, double* value);
```

Descrição

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

Parâmetros

`handle`: ponteiro para um `ContainerHandle`.

`name`: string que contém o nome da propriedade a ser lida.

`value`: ponteiro para variável double onde será escrito o valor da propriedade.

Retorno

CONTAINER_SUCCESS caso o valor da variável seja lido, CONTAINER_ERROR_INVALID_PROPERTY caso a propriedade não exista ou não seja do tipo double.

---

containerSetCharProperty

Protótipo da Função

```
int containerSetCharProperty(ContainerHandle* handle, const char* name, char value);
```

Descrição

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

Parâmetros

`handle`: ponteiro para um `ContainerHandle`.

`name`: string que contém o nome da propriedade a ser alterada.

`value`: valor que deverá ser atribuido à propriedade.

Retorno

CONTAINER_SUCCESS caso o valor da variável seja alterado, CONTAINER_ERROR_INVALID_PROPERTY caso a propriedade não exista ou não seja do tipo char.

---

containerGetCharProperty

Protótipo da Função

```
int containerGetCharProperty(ContainerHandle* handle, const char* name, char* value);
```

Descrição

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

Parâmetros

`handle`: ponteiro para um `ContainerHandle`.

`name`: string que contém o nome da propriedade a ser lida.

`value`: ponteiro para variável char onde será escrito o valor da propriedade.

Retorno

CONTAINER_SUCCESS caso o valor da variável seja lido, CONTAINER_ERROR_INVALID_PROPERTY caso a propriedade não exista ou não seja do tipo char.

---

containerFindFirst

Protótipo da Função

```
int containerFindFirst(ContainerHandle* handle, ContainerImage* image, ContainerResultList* list);
```

Descrição

Reconhece o código e o guarda em um objeto `ContainerRecognition` que se encontra no primeiro nó do `ContainerResultList`. A imagem deverá ser carregada utilizando as funções `containerLoadImage` ou `containerLoadImageFromMemory`. Caso não seja encontrado nenhum código, ou caso o hardkey não esteja autorizado ou não foi encontrado, o objeto `ContainerRecognition` conterá uma string vazia como código.

Esta função deverá ser chamada apenas com um `ContainerResultList` vazio.

Parâmetros

`handle`: ponteiro para um `ContainerHandle` que contém a configuração da biblioteca.

`image`: ponteiro para um `ContainerImage` que contém a imagem a ser processada.

`list`: ponteiro para um `ContainerResultList` onde será armazenado o resultado do processamento.

Retorno

CONTAINER_SUCCESS caso a imagem seja processada, caso contrário um outro valor do enum `containerError`.

---

containerFindNext

Protótipo da Função

```
int containerFindNext(ContainerHandle* handle, ContainerImage* image, ContainerResultList* list);
```

Descrição

O objetivo desta função é permitir ao usuário reconhecer múltiplos códigos em uma mesma imagem. A função reconhece um código e o guarda em um objeto `ContainerRecognition` que se encontra no último nó do `ContainerResultList`. A imagem deverá ser carregada utilizando as funções `containerLoadImage` ou `containerLoadImageFromMemory`. Caso não seja encontrado nenhum código, ou caso o hardkey não esteja autorizado ou não foi encontrado, o objeto `ContainerRecognition` conterá uma string vazia como código.

Está função deverá ser chamada apenas com um `ContainerResultList` anteriormente processado pela função `containerFindFirst` ou `containerFindNext`.

Parâmetros

`handle`: ponteiro para um `ContainerHandle` que contém a configuração da biblioteca.

`image`: ponteiro para um `ContainerImage` que contém a imagem a ser processada.

`list`: ponteiro para um `ContainerResultList` onde será armazenado o resultado do processamento.

Retorno

CONTAINER_SUCCESS caso a imagem seja processada, caso contrário um outro valor do enum `containerError`.

---

containerLoadImage

Protótipo da Função

```
int containerLoadImage(const char* filename, ContainerImage** img);
```

Descrição

Carrega uma imagem de um arquivo e salva a referência como um ContainerImage.

O arquivo de imagem deverá ser um bitmap, jpeg ou png.

Parâmetros

`filename`: path para o arquivo da imagem.

`img`: ponteiro-para-ponteiro para o struct `ContainerImage` onde será armazenada a imagem.

Retorno

CONTAINER_SUCCESS caso a imagem seja carregada corretamente, CONTAINER_ERROR_FILE_NOT_FOUND caso o arquivo não seja encontrado ou não exista, CONTAINER_ERROR_INVALID_IMAGE ou CONTAINER_ERROR_INVALID_IMAGE_TYPE em caso de problemas na carga da imagem.

---

containerLoadImageFromMemory

Protótipo da Função

```
int containerLoadImageFromMemory(const unsigned char* buf, int n, int type, int width, int height, ContainerImage** img);
```

Descrição

Carrega uma imagem de um array de bytes e salva a referência como um `ContainerImage`.

A imagem deve estar em algum formato estruturado (bmp, jpg, png e etc.) ou raw (Grayscale 8bit, RGB ou BGR).

Parâmetros

`buf`: array de bytes contendo a imagem.

`n`: tamanho do array em bytes.

`type`: tipo da imagem:

- tipos estruturados=0, GRAY8=1, RGB=2, BGR=3.

`width`: largura da imagem, ignorado se type==0.

`height`: altura da imagem, ignorado se type==0.

`img`: ponteiro-para-ponteiro para um `ContainerImage` onde será armazenada a imagem.

Retorno

CONTAINER_SUCCESS caso a imagem seja carregada corretamente, CONTAINER_ERROR_FILE_NOT_FOUND caso o arquivo não seja encontrado ou não exista, CONTAINER_ERROR_INVALID_IMAGE ou CONTAINER_ERROR_INVALID_IMAGE_TYPE em caso de problemas na carga da imagem.

---

containerFreeImage

Protótipo da Função

```
int containerFreeImage(ContainerImage** img);
```

Descrição

Libera a memória alocada para armazenar uma imagem.

Parâmetros

`img`: ponteiro-para-ponteiro para um `ContainerImage` que será desalocado.

Retorno

Sempre retorna CONTAINER_SUCCESS.

containerBuildInfo

Protótipo da Função

```
const char* containerBuildInfo();
```

Descrição

Verifica as informações de build da biblioteca, sendo utilizada para verificar se a versão que está sendo executada é a esperada.

Parâmetros

Não possui

Retorno

String constante que possui 12 ou 13 caracteres que representam o BuildInfo além de um terminador ('\0').

containerNumThreads

Protótipo da Função

```
int containerNumThreads();
```

Descrição

Verifica o número de threads autorizadas no hardkey.

Parâmetros

Não possui

Retorno

Número inteiro que representa quantas threads estão autorizadas a executarem simultaneamente as funções de OCR da biblioteca. Retorna 1 caso o hardkey não seja encontrado.

4.1.2.3 API 2 - Configuração

Nesta seção detalhamos todos os parâmetros de configuração disponíveis na API 2. Os parâmetros, tipos e valores são os mesmos para qualquer linguagem.

int timeout

Descrição

Após `timeout` milisegundos desde o início de processamento de uma imagem a busca do código será encerrada e será retornado o melhor código encontrado. Caso `timeout` seja zero, não há timeout. Recomenda-se utilizar um `timeout` diferente de zero quando a aplicação exige que as imagens sejam processadas rapidamente (com baixa latência) ou quando a carga da CPU está muito elevada.

O valor **default** é **0**.

int flip

Descrição

Serve para definir a orientação do código que será lido na imagem. Em alguns casos os códigos podem estar invertidos (rotacionados de 180 graus) ou podem aparecer em ambas as orientações.

A recomendação é que se utilize o valor 0 para imagens laterais ou das portas, 1 para laterais ou portas onde a câmera se encontra rotacionada de 180 graus e 2 para imagens de topo onde o código pode aparecer em duas orientações distintas.

Os valores são:

- CONTAINER_FLIP_0 == 0 (**default**)
- CONTAINER_FLIP_180 == 1
- CONTAINER_FLIP_AMBOS == 2

`int secondBest`

Descrição

A opção secondBest pode estar habilitada (1) ou desabilitada (0).

Quando um código é localizado porém o dígito verificador lido não está de acordo com o valor esperado (calculado), a biblioteca internamente utiliza a próxima opção lida para o caractere com a probabilidade mais baixa. Desta forma, caso o verificador esteja de acordo, a leitura é retornada.

O valor **default** é **1**.

`int partialRecognition`

Descrição

A opção partialRecognition pode estar habilitada (1) ou desabilitada (0).

Quando um código é localizado porém existe algum caractere que está com probabilidade de acerto abaixo de `minProbPerCharacter` o código apenas será retornado se esta opção estiver habilitada e o caractere será substituido por `lowProbabilityChar`.

Está opção é útil quando se deseja efetuar a leitura de códigos com muitas avarias, porém o índice de falsos positivos poderá aumentar já que outros textos podem acabar lidos como códigos parciais.

O valor **default** é **0**.

`int ignoreBorders`

Descrição

A opção ignoreBorders pode estar habilitada (1) ou desabilitada (0).

Quando um código é localizado a menos de 2px de distância da borda da imagem, ele será descartado se a opção ignoreBorders estiver habilitada. Normalmente é útil quando se deseja identificar o código de veículos em movimento para evitar que uma letra que ainda não está totalmente na imagem seja reconhecida erroneamente, por exemplo uma letra 'H' pode acabar parecendo um 'I' caso ainda não esteja completamente visível.

O valor **default** é **1**.

`double minProbPerCharacter`

Descrição

Probabilidade (confiabilidade) mínima exigida no reconhecimento de cada caractere. É extremamente importante para o bom funcionamento do OCR, e não recomenda-se mudar a configuração default. No entanto, em casos específicos pode ser interessante ajustá-lo.

Se `minProbPerCharacter` for menor que o default, o número de códigos que não são reconhecidos reduzirá, mas em contrapartida o número de códigos com algum caractere errado poderá aumentar.

Se `minProbPerCharacter` for maior que o default, o número de códigos que não são reconhecidos poderá aumentar, mas o número de erros será menor.

O valor **default** é **0.2**.

`char lowProbabilityChar`

Descrição

Caractere de substituição a ser utilizado quando um caractere do código é reconhecido com probabilidade menor que `minProbPerCharacter`. Terá efeito apenas se a opção `partialRecognition` estiver habilitada (1).

Por exemplo, se `lowProbabilityChar='-'` e `partialRecognition=1`, o código `"ABCD1234567"` será retornado como `"A-CD1234567"` se a probabilidade do segundo caractere for menor que `minProbPerCharacter`.

O valor **default** é **'-'**.

`int minCharHeight`

Descrição

Altura mínima que um caractere deve ter, em pixels.

O valor **default** é **9**.

```
int maxCharHeight
```

Descrição

Altura máxima que um caractere deve ter, em pixels.

O valor **default** é **200**.

4.2. Exemplos de utilização

4.2.1 Exemplo C/C++ (API1)

```c
#include <stdio.h>
    #include "containerCore.h"

    int main(int argc, char* argv[])
    {
        ContainerRecognition rec;
        ContainerConfig config;
        config.timeout = 0;
        config.flip = CONTAINER_FLIP_0;
        containerReadText(argv[1], &config, &rec);
        printf("text: %s\n", rec.text);
        return 0;
    }
```

4.2.2 Exemplo C/C++ (API2)

```c
#include <stdio.h>
    #include "containerCore.h"

    int main(int argc, char* argv[])
    {
        int res = 0;

        ContainerImage* img;
        ContainerResultList resultList;
        ContainerRecognition* rec;
        ContainerHandle* handle = containerInit();

        res = containerSetIntProperty(handle, "timeout", 0);
        /* tratar o retorno aqui */

        res = containerSetIntProperty(handle, "flip", CONTAINER_FLIP_0);
        /* tratar o retorno aqui */

        res = containerLoadImage(argv[1], &img);
        /* tratar o retorno aqui */

        res = containerFindFirst(handle, img, &resultList);
        rec = resultList.recognition;
        /* tratar o retorno aqui */

        printf("text: %s\n", rec->text);

        res = containerFreeResultList(&resultList);
        /* tratar o retorno aqui */

        res = containerFreeImage(&img);
        /* tratar o retorno aqui */

        return 0;
    }
```

API Index

C/C++

