



PUMATRONIX

movimento em foco

Jidosha Rail

Manual do Usuário

Biblioteca de software para reconhecimento de códigos de vagões padrão RFFSAs

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

Table of Contents

- **Releases**
- 1. Visão Geral
 - 1.1. Condições Gerais
 - 1.2. Licença de software
- 2. Introdução
 - 2.1. Objetivo
 - 2.2. Condições de Uso
- 3. Instalação
 - 3.1. Windows
 - 3.2. Linux
 - 3.2.1. Configuração das permissões *dohardkey*
- 4. API
 - 4.1. API JIDOSHA RAIL (C/C++)
 - 4.2. Exemplos de utilização
 - 4.2.1 Exemplo C/C++ (API1)
 - 4.2.2 Exemplo C/C++ (API2)

Releases

Date	Version	Revision
29/12/2020	2.0.0	<ul style="list-style-type: none">• 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.

Copyright © Pumatronix Equipamentos Eletrônicos Ltda. Todos os direitos reservados.

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 RAIL** tem por objetivo detalhar as funções da biblioteca de software especializada em reconhecimento de códigos de vagões padrão RFFSA, chamada JIDOSHA RAIL, e suas condições de uso para correto funcionamento.

2.1. Objetivo

A biblioteca de software JIDOSHA RAIL tem como principal funcionalidade reconhecer códigos de vagões no padrão RFFSA a partir de imagens. Sua principal aplicação é na gestão e controle de acesso aos portos ferroviários, 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 manobras ferroviários.

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

2.2. Condições de Uso

A biblioteca de software JIDOSHA RAIL 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 biblioteca 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 RAIL 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 RAIL, 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 "railSample.exe 0 0 Rail_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 RAIL (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, railCore.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 RAIL, tem como principal motivação a facilidade de uso. É possível ler códigos através de uma única chamada de função ([railReadText](#) ou [railReadTextFromMemory](#), 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 RAIL para uma versão mais recente, sem precisar recompilar).

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

railCore.h

```
#define MAX_NUM_CHARS 11

#define RailHandle void
#define RailImage void

enum railFlip {
    RAIL_FLIP_0 = 0,
    RAIL_FLIP_180 = 1,
    RAIL_FLIP_AMBOS = 2,
};

enum railError {
    RAIL_SUCCESS = 0,
    RAIL_ERROR_HARDKEY_NOT_FOUND = 1,
    RAIL_ERROR_HARDKEY_NOT_AUTHORIZED = 2,
    RAIL_ERROR_FILE_NOT_FOUND = 3,
    RAIL_ERROR_INVALID_IMAGE = 4,
    RAIL_ERROR_INVALID_IMAGE_TYPE = 5,
    RAIL_ERROR_INVALID_PROPERTY = 6,
    RAIL_ERROR_COUNTRY_NOT_SUPPORTED = 7,
    RAIL_ERROR_OTHER = 999,
};

enum railImageType {
    RAIL_IMAGE_TYPE_STRUCTURED = 0,
    RAIL_IMAGE_TYPE_RAW_GRAY8 = 1,
    RAIL_IMAGE_TYPE_RAW_RGB = 2,
    RAIL_IMAGE_TYPE_RAW_BGR = 3,
};

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

typedef struct RailRecognition
{
    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;
} RailRecognition;

typedef struct RailResultList
{
    struct RailResultList* next;
    struct RailRecognition* recognition;
} RailResultList;

JIDOSHACORE_API int railReadTextFromMemory(const unsigned char* stream, int n, RailConfig* config, RailRecognition* rec);
JIDOSHACORE_API int railReadText(const char* filename, RailConfig* config, RailRecognition* rec);
JIDOSHACORE_API int railGetVersion(int* major, int* minor, int* release);
JIDOSHACORE_API int railGetHardkeySerial(unsigned long* serial);
JIDOSHACORE_API int railGetHardkeyState(int* state);

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

JIDOSHACORE_API const char* railBuildInfo();
JIDOSHACORE_API int railNumThreads();
```

```
// API 2
JIDOSHACORE_API RailHandle* railInit();
JIDOSHACORE_API int railDestroy(RailHandle* handle);
JIDOSHACORE_API int railLoadImage(const char* filename, RailImage** img);
JIDOSHACORE_API int railLoadImageFromMemory(unsigned char* stream, int length, int type, int width, int height, RailImage** img);
JIDOSHACORE_API int railFreeImage(RailImage** img);
JIDOSHACORE_API int railSetIntProperty(RailHandle* handle, const char* name, int value);
JIDOSHACORE_API int railGetIntProperty(RailHandle* handle, const char* name, int* value);
JIDOSHACORE_API int railSetDoubleProperty(RailHandle* handle, const char* name, double value);
JIDOSHACORE_API int railGetDoubleProperty(RailHandle* handle, const char* name, double* value);
JIDOSHACORE_API int railSetCharProperty(RailHandle* handle, const char* name, char value);
JIDOSHACORE_API int railGetCharProperty(RailHandle* handle, const char* name, char* value);
JIDOSHACORE_API int railFreeResultList(RailResultList* list);
JIDOSHACORE_API int railFindFirst(RailHandle* handle, RailImage* img, RailResultList* list);
JIDOSHACORE_API int railFindNext(RailHandle* handle, RailImage* img, RailResultList* list);
```

4.1.1. API 1

4.1.1.1 Structs

struct RailConfig

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 milissegundos. 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:

RAIL_FLIP_0: O código está na orientação natural da imagem.

RAIL_FLIP_180: O código está invertido com relação à imagem. Recomendado para casos onde a câmera está instalada rotacionada em 180 graus.

RAIL_FLIP_AMBOS: O código pode estar normal ou invertido. Esta opção existe apenas para compatibilidade com o produto JIDOSHA CONTAINER.

struct RailRecognition

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

railReadText

Protótipo da Função


```
int railReadText(const char* filename, RailConfig* config, RailRecognition* rec);
```

Descrição

Reconhece o código na imagem e o guarda em um objeto [RailRecognition](#). 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 [RailRecognition](#) conterà 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 [RailConfig](#) com a configuração para a biblioteca.

rec: ponteiro para a struct [RailRecognition](#) onde será armazenado o resultado da leitura.

Retorno

Código de erro: RAIL_SUCCESS no caso de sucesso, algum valor de railError caso contrário.

railReadTextFromMemory

Protótipo da Função

```
int railReadTextFromMemory(const unsigned char* stream, int n, RailConfig* config, RailRecognition* rec);
```

Descrição

Reconhece o código na imagem e o guarda em um objeto [RailRecognition](#). 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 [RailRecognition](#) conterà 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 [RailConfig](#) com a configuração para a biblioteca.

rec: ponteiro para a struct [RailRecognition](#) onde será armazenado o resultado da leitura.

Retorno

Código de erro: RAIL_SUCCESS no caso de sucesso, algum valor de railError caso contrário.

railGetVersion

Protótipo da Função

```
int railGetVersion(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 RAIL_SUCCESS.

railGetHardkeySerial

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

RAIL_SUCCESS em caso de sucesso, RAIL_ERROR_HARDKEY_NOT_FOUND caso o hardkey não tenha sido encontrado.

railGetHardkeyState

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

RAIL_SUCCESS em caso de sucesso, RAIL_ERROR_HARDKEY_NOT_FOUND caso o hardkey não tenha sido encontrado.

railGetHardkeyRemainingTime

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

RAIL_SUCCESS em caso de sucesso, RAIL_ERROR_HARDKEY_NOT_FOUND caso o hardkey não tenha sido encontrado.

4.1.2. API 2

4.1.2.1 Structs

struct RailResultList

Descrição

A finalidade dessa estrutura é armazenar a lista encadeada com os resultados dos processamentos das funções [railFindFirst](#) e [railFindNext](#).

Membros

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

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

typedef void RailHandle

Descrição

Tipo utilizado para representar a memória alocada para a configuração.

typedef void RailImage

Descrição

Tipo utilizado para representar a memória alocada para uma imagem.

4.1.2.2 Funções

railFreeResultList

Protótipo da Função

```
void railFreeResultList(RailResultList* list);
```

Descrição

Libera a memória alocada para lista encadeada de resultados.

Parâmetros

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

Retorno

Não possui retorno.

railInit

Protótipo da Função

```
RailHandle* railInit();
```

Descrição

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

Parâmetros

Não possui.

Retorno

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

railDestroy

Protótipo da Função

```
int railDestroy(RailHandle* handle);
```

Descrição

Libera a memória alocada pela função `railInit`.

Parâmetros

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

Retorno

Sempre retorna RAIL_SUCCESS.

railSetIntProperty

Protótipo da Função

```
int railSetIntProperty(RailHandle* handle, const char* name, int value);
```

Descrição

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

Parâmetros

handle: ponteiro para um [RailHandle](#).

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

value: valor que deverá ser atribuído à propriedade.

Retorno

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

railGetIntProperty

Protótipo da Função

```
int railGetIntProperty(RailHandle* handle, const char* name, int* value);
```

Descrição

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

Parâmetros

handle: ponteiro para um [RailHandle](#).

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

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

Retorno

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

railSetDoubleProperty

Protótipo da Função

```
int railSetDoubleProperty(RailHandle* handle, const char* name, double value);
```

Descrição

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

Parâmetros

handle: ponteiro para um [RailHandle](#).

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

value: valor que deverá ser atribuído à propriedade.

Retorno

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

railGetDoubleProperty

Protótipo da Função

```
int railGetDoubleProperty(RailHandle* handle, const char* name, double* value);
```

Descrição

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

Parâmetros

handle: ponteiro para um [RailHandle](#).

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

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

Retorno

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

railSetCharProperty

Protótipo da Função

```
int railSetCharProperty(RailHandle* handle, const char* name, char value);
```

Descrição

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

Parâmetros

handle: ponteiro para um [RailHandle](#).

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

value: valor que deverá ser atribuído à propriedade.

Retorno

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

railGetCharProperty

Protótipo da Função

```
int railGetCharProperty(RailHandle* handle, const char* name, char* value);
```

Descrição

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

Parâmetros

handle: ponteiro para um [RailHandle](#).

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

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

Retorno

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

railFindFirst

Protótipo da Função

```
int railFindFirst(RailHandle* handle, RailImage* image, RailResultList* list);
```

Descrição

Reconhece o código e o guarda em um objeto [RailRecognition](#) que se encontra no primeiro nó do [RailResultList](#). A imagem deverá ser carregada utilizando as funções [railLoadImage](#) ou [railLoadImageFromMemory](#). Caso não seja encontrado nenhum código, ou caso o hardkey não esteja autorizado ou não foi encontrado, o objeto [RailRecognition](#) conterá uma string vazia como código.

Esta função deverá ser chamada apenas com um [RailResultList](#) vazio.

Parâmetros

[handle](#): ponteiro para um [RailHandle](#) que contém a configuração da biblioteca.

[image](#): ponteiro para um [RailImage](#) que contém a imagem a ser processada.

[list](#): ponteiro para um [RailResultList](#) onde será armazenado o resultado do processamento.

Retorno

RAIL_SUCCESS caso a imagem seja processada, caso contrário um outro valor do enum [railError](#).

railFindNext

Protótipo da Função

```
int railFindNext(RailHandle* handle, RailImage* image, RailResultList* 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 [RailRecognition](#) que se encontra no último nó do [RailResultList](#). A imagem deverá ser carregada utilizando as funções [railLoadImage](#) ou [railLoadImageFromMemory](#). Caso não seja encontrado nenhum código, ou caso o hardkey não esteja autorizado ou não foi encontrado, o objeto [RailRecognition](#) conterá uma string vazia como código.

Esta função deverá ser chamada apenas com um [RailResultList](#) anteriormente processado pela função [railFindFirst](#) ou [railFindNext](#).

Parâmetros

[handle](#): ponteiro para um [RailHandle](#) que contém a configuração da biblioteca.

[image](#): ponteiro para um [RailImage](#) que contém a imagem a ser processada.

[list](#): ponteiro para um [RailResultList](#) onde será armazenado o resultado do processamento.

Retorno

RAIL_SUCCESS caso a imagem seja processada, caso contrário um outro valor do enum [railError](#).

railLoadImage

Protótipo da Função

```
int railLoadImage(const char* filename, RailImage** img);
```

Descrição

Carrega uma imagem de um arquivo e salva a referência como um [RailImage](#).

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 [RailImage](#) onde será armazenada a imagem.

Retorno

RAIL_SUCCESS caso a imagem seja carregada corretamente, RAIL_ERROR_FILE_NOT_FOUND caso o arquivo não seja encontrado ou não exista, RAIL_ERROR_INVALID_IMAGE ou RAIL_ERROR_INVALID_IMAGE_TYPE em caso de problemas na carga da imagem.

railLoadImageFromMemory

Protótipo da Função

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

Descrição

Carrega uma imagem de um array de bytes e salva a referência como um [RailImage](#).

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 [RailImage](#) onde será armazenada a imagem.

Retorno

RAIL_SUCCESS caso a imagem seja carregada corretamente, RAIL_ERROR_FILE_NOT_FOUND caso o arquivo não seja encontrado ou não exista, RAIL_ERROR_INVALID_IMAGE ou RAIL_ERROR_INVALID_IMAGE_TYPE em caso de problemas na carga da imagem.

railFreeImage

Protótipo da Função

```
int railFreeImage(RailImage** img);
```

Descrição

Libera a memória alocada para armazenar uma imagem.

Parâmetros

img: ponteiro-para-ponteiro para um [RailImage](#) que será desalocado.

Retorno

Sempre retorna RAIL_SUCCESS.

railBuildInfo

Protótipo da Função

```
const char* railBuildInfo();
```

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').

railNumThreads

Protótipo da Função

```
int railNumThreads();
```

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` milissegundos 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 em orientação normal, e 1 para imagens rotacionadas em 180 graus.

Os valores são:

- RAIL_FLIP_0 == 0 (**default**)
- RAIL_FLIP_180 == 1
- RAIL_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 substituído por `lowProbabilityChar`.

Esta 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 "ABC1234567" será retornado como "A-C1234567" 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)

```
#include <stdio.h>
#include "railCore.h"

int main(int argc, char* argv[])
{
    RailRecognition rec;
    RailConfig config;
    config.timeout = 0;
    config.flip = RAIL_FLIP_0;
    railReadText(argv[1], &config, &rec);
    printf("text: %s\n", rec.text);
    return 0;
}
```

4.2.2 Exemplo C/C++ (API2)

```
#include <stdio.h>
#include "railCore.h"

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

    RailImage* img;
    RailResultList resultList;
    RailRecognition* rec;
    RailHandle* handle = railInit();

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

    res = railSetIntProperty(handle, "flip", RAIL_FLIP_0);
    /* tratar o retorno aqui */

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

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

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

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

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

    return 0;
}
```

API Index

C/C++

- [char lowProbabilityChar](#)
- [double minProbPerCharacter](#)
- [int flip](#)
- [int ignoreBorders](#)
- [int maxCharHeight](#)
- [int minCharHeight](#)
- [int partialRecognition](#)
- [int secondBest](#)
- [int timeout](#)
- [railBuildInfo](#)
- [railDestroy](#)
- [railFindFirst](#)
- [railFindNext](#)
- [railFreeImage](#)
- [railFreeResultList](#)
- [railGetCharProperty](#)
- [railGetDoubleProperty](#)
- [railGetHardkeyRemainingTime](#)
- [railGetHardkeySerial](#)
- [railGetHardkeyState](#)
- [railGetIntProperty](#)
- [railGetVersion](#)
- [railInit](#)
- [railLoadImage](#)
- [railLoadImageFromMemory](#)
- [railNumThreads](#)
- [railReadText](#)
- [railReadTextFromMemory](#)
- [railSetCharProperty](#)
- [railSetDoubleProperty](#)
- [railSetIntProperty](#)
- [struct RailConfig](#)
- [struct RailRecognition](#)
- [struct RailResultList](#)
- [typedef void RailHandle](#)
- [typedef void RailImage](#)