

Acessando DLL'S no ruby (exemplo com ECF)

Ruby realmente é mais do que simplesmente programar para a web !

Eu mesmo somente decidi migrar todos os aplicativos da Object Data para Ruby em razão da linguagem ser de aplicação geral.

Nesse artigo mostrarei como acessar bibliotecas ".dll" e ".so" através da DL, biblioteca da Standard API.


Mostraremos como criar um ".so" no linux e no windows acessar um Emissor de Cupom Fiscal - ECF.

Conhecendo as API's de acesso a DLL

Inicialmente devemos compreender que o acesso a DLL'S em Ruby é feito através de duas API's. A primeira é a Win32API e a segunda é a DL.

A primeira como o próprio nome diz é usada exclusivamente em ambiente Windows, já a segunda é multi-plataforma. A única diferença, nesse último caso, é que em Windows a extensão ficará ".dll" e em Linux e outros Unix ".so".

Ambas fazem parte da Standard API e podem ser acessadas no site [ruby-doc.org](http://www.ruby-doc.org).



Neste artigo iremos utilizar a DL por ser multi-plataforma.

Inicialmente, vamos criar nossa biblioteca em C chamada `hellolib.c`

nosso código em C

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

```
int imprimir()
```

```
{  
    return printf("hello world ! \n");  
}
```

```
int somar(int a, int b)
```

```
{
```

```
return a + b;
}
```

Compilando nossa ".so"

```
gcc hellolib.c -shared -o hellolib.so
```

Acessando através do ruby:

```
irb
irb(main):001:0> require 'dl'
=> true
irb(main):002:0> handle = DL.dlopen('./hellolib.so')
=> #<DL::Handle:0xb7ef1ad4>
irb(main):003:0> result, params = handle['somar', 'III'].call(3, 5)
=> [8, [3, 5]]
irb(main):004:0> result
=> 8
irb(main):005:0> params
=> [3, 5]
```

Onde 'III' é o retorno mais os parâmetros da dll. Nesse caso se espera um inteiro de retorno e são passados dois inteiros como argumento.

```
irb(main):006:0> result, params = handle['imprimir', 'I'].call
hello world !
=> [15, []]
```

Agora temos uma impressão no prompt e apenas um 'I' indicando que o valor de retorno deve ser um inteiro.

Efetuada uma leitura X de um 'ECF'

Utilizando DL agora no windows, podemos imprimir uma leitura x de uma impressora fiscal.

Nessa caso estarei usando uma impressora fiscal Elgin veja:

```
require 'dl'
# Elgin.dll está em c:/windows/system32
handle = DL.dlopen('Elgin.dll')
handle['Elgin_LeituraX', 'I'].call
```

O retorno da impressora será um número inteiro que pode ser :

0: indica erro na execução da função. 1: indica que nenhum erro ocorreu -4: O arquivo de inicialização Elgin.ini não foi encontrado no diretório de sistema do Windows. -5: Erro ao abrir a porta de comunicação. -27: Status da impressora diferente de 6,0,0 (ACK, ST1 e ST2). -50: Número de série inválido

Os parâmetros quando são enviados, podem ser captadores no seu retorno.

```
result, params = handle[...]
```

Assim, se houverem alterações nos valores podemos saber seu retorno.

Um exemplo disso é quando precisamos capturar o estado de uma impressora fiscal. O valor de retorno nos informa se o procedimento foi efetuado com sucesso, erro ao executar a função, mas não o estado interno da impressora, veja a documentação a seguir:

Elgin_VerificaEstadoImpressora

Retorna o estado da impressora.

Parâmetros:

ACK: Variável inteira para receber o primeiro byte.

ST1: Variável inteira para receber o segundo byte.

ST2: Variável inteira para receber o terceiro byte.

Possíveis retornos da Função (INTEIRO):

0: indica erro na execução da função.

1: indica que nenhum erro ocorreu.

-1: Erro de execução da função.

-4: O arquivo de inicialização Elgin.ini não foi encontrado no diretório de sistema do Windows.

-5: Erro ao abrir a porta de comunicação.

-8: Erro ao criar ou gravar no arquivo STATUS.TXT ou RETORNO.TXT.

-27: Status da impressora diferente de 6,0,0 (ACK, ST1 e ST2).

-50: Número de série inválido.

Agora executando em ruby:

```
#em ".call" sao espaços em branco :)
```

```
result, params = execute('Elgin_VerificaEstadoImpressora', 'ISSS').call(' ', ' ', ' ')
```

Verifique que passei uma String com um espaço em branco em cada parâmetro para receber o valor. Com o método unpack de String posso fazer duas coisas:

1) Verificar o valor como um inteiro:
st1 = params[1].unpack('C').to_s.to_i
puts "pouco papel..." if st1 >= 64

2) Verificando os bytes:
puts params[1].unpack('B8').to_s
"01000000"

no exemplo acima verificamos que o valor é 64 pois o sétimo bit está setado sendo:

1 => 1
2 => 2
3 => 4
4 => 8
5 => 16
6 => 32
7 => 64
8 => 128

Ok, espero ter ajudado !

Bom código,

Alexandre Riveira