Reverse Polish Notation

Secure your Mobile software

by Protectedvoid.net

This document describes the RPN library developed by protectedvoid.net. This is done in a functional way, as well as a technical way.

Introduction

When you have developed your application for the Windows Mobile platform, you probably don't want everyone to use this product for free. You want to prevent customers from handing your software over to friends for nothing. You will definitely earn more money when you protect your software from piracy. Doing this based on RPN calculations does not make your software bullet proof, but it is a standard that is supported by almost all distributors (pocketland, pocketgear, handango, mobihand, ...) and supports a pretty good level of protection. How this process works, will be described in the following chapters.

This library will support you protecting your software using RPN in a various ways. All different features of this library will be explained in next chapters, as well as a technical description and the operator support (which is pretty much everything) and some code samples.

Requirements

The library is developed using the **Microsoft. NET Compact Framework 3.5** and written in C#. Therefore this framework is required to be used on devices.

Features

- Activation code generation
    - Based on the owner name of the device and you RPN string, the library generates the code, which must be matched with the code supplied by the customer.
- Activation code storage
    - This library supports a `Storage` class which you can use from your application. If this class is used, the activation code that the user fills in your registration window, will automatically saved in the registry. You can retrieve this code through this class also. For a more technical description of this feature, please check the following chapters.
- Automatic activation
    - This library comes with a base Form class (`MainFormBase`), from which you can derive your main form in your application. If you do this, all required events are picked up and the existence of a valid registration code is checked. If this is not the case, the built in Activation window will popup, asking the user for a valid code. Using this feature, you hardly have to worry about activation and can focus on the main purpose of your application.
- Easy owner information retrieval
    - Another available class is the `DeviceOwner` class. Using this class you have easy access to the owner's name, phone number, notes and email address. The name of this class is the name of the owner, which will be used for the code generation and validation. No need to specify this, it is retrieved automatically from within the calculator.
- Test tool
    - For easily testing of RPN strings, an windows (desktop) based test utility is also delivered with this library.

Technical overview

The main class of the library is the `Calculator`. This calculator must be instantiated with an RPN string, and optionally a `Storage` class. This storage class describes your vendor name and the product name. Based on this information, the calculator can retrieve the customer's activation code from the registry. If this class is not specified, you cannot use this automatic validation feature. The location of the history is "HKLM\Software\{VendorName}\ProductCode". A string value will be created here with the name "RegistrationCode".

Operator support

The following operators are supported:

- Logical operators
  - && (Logical And): Checks if both values are true
  - || (Logical Or): Checks if one of both values is true
  - ! (Not): Checks if one value is false
  - == (Equals): Checks if two values are equal
  - < (Less): Checks if value1 is less than value2
  - <= (Less or equal): Checks if value1 is less or equal to value2
  - > (Greater): Checks if value1 is larger than value2
  - >= (Greater or equal): Checks if value1 is larger of equal to value2
  - != (Not equal): Checks if two values are not equal
- Bitwise operators:
  - << (Shift left): Performs 1 binary shift left on a single value
  - >> (Shift right): Performs 1 binary shift right on a single value
  - ~ (Invert): Inverts a single value
  - & (Bitwise And): Performs an bitwise and on two values
  - | (Bitwise Or): Performs an bitwise or on two values
- Arithmetic operators
  - + (Add): Adds value1 to value2
  - - (Subtract): Subtracts value2 from value1
  - * (Multiply): Multiplies value1 by value2
  - / (Divide): Divides value1 by value 2
  - % (Modulo): Finds the remained of division of value1 by value2

Code Samples

This chapter describes some samples for using this library. Code samples are written in C#, but converting them to VB.NET is trivial.

Calculate an activation code based on a name and RPN String

```
// Create a new instance of the Calculator class
Calculator calculator = new Calculator("i c + 12 *");
// Generate a code based on the name of the owner
string code = calculator.GenerateCode(DeviceOwner.Instance.Name);
```

Generate and validate a code using the Storage class and automatic retrieval of owner name

```
// Create a new Storage class based on the vendor and application name
Storage storage = new Storage("Protectedvoid", "MyApplication");
storage.RegistrationCode = userActivateCode; // The code which has been
entered by the user

// Create a new instance of the Calculator class
Calculator calculator = new Calculator("i c + 12 *", storage);
// Generate a code based on the owner of the device and the RPN string
// and validate it ahainst the code from te registry
bool hasValidCode = calculator.ValidateCode();
```

Inherit from the base Form and implement automatic activation

```
internal partial class MyMainApplicationForm : MainFormBase
{
    // Supply the base class with the vendor name
    protected override string VendorName
    {
        get { return "Protectedvoid"; }
    }

    // Supply the base class with the product name
    protected override string ProductName
    {
        get { return "EasyShopping"; }
    }

    // Supply the base class with a rpn string
    protected override string RpnString
    {
        get { return "i c * 2 * key + c 6 * + 3 - i *"; }
    }

    // Supply the base class with a message in case there is no valid
    // owner specified on the device
    protected override string NoOwnerMessage
    {
        get { return resourceMessageNoOwner.Text; }
    }

    // The rest of your forms implementation will be here
}
```

By using this implementation, the base class (`MainFormBase`) will hook into the Activated and Load event to check for the existence of a code in the registry based on the supplied Vendor and product name. A built in activation window will be show, but you can also create your own registration screen and supply that to the base form using the `OnActivationWindowRequired` override.

Conclusion

As you can see you have several ways to use this library. I would suggest to implement something alike the last sample. This way you don't need to bother about the customers registration key and the complete process around. Focus on your application and make money!

Questions

For questions, remarks or anything, please feel free to contact me at:

support@protectedvoid.net