ZINCKSOFT

Give The Dream New Life.

JSU.PassMeter

The complete Module API Reference

Quick Introduction 1
Including the module in your page 1
JSU.PassRule API Reference 2
Constructor 2
Properties 2
__func__ 2
Methods 3
getScore 3
Defaults 3
defaultRules 3
JSU.PassMeter API Reference 4
Constructor 4
Properties 4
__rules__ 4
__intpt__ 4
Methods 4
changeRules 4
changeInterpretation 5
getStrength 5
Defaults 5
defaultInterpretation 5
The JSU.PassMeter in action 6
Creating a custom strength meter 7

Quick Introduction

The JSU PassMeter Module (**JSU.PassMeter / JSUPassMeter**) offers you an extensible object-oriented password strength meter, allowing you to easily check the strength of user passwords, or create your own measuring algorithm for your custom needs.

Including the module in your page

Before including this module in your page, first include the core module:

```
<script type="text/javascript" src="jsu.core.js"></script>

<script type="text/javascript" src="jsu.password.js"></script>
```

JSU.PassRule API Reference

A. Constructor

Syntax:

```
JSU.PassRule(rule)
```

Arguments:

- ***rule***: a function that returns a score obtained by executing it on a custom password; example of rules could be checking if the password contains only letters, or checking if the password contains both letters and numbers which will produce a higher score than the first example rule.

Returns: Must return a numeric value used to cumulate the scores of each rule applied.

Example:

```
new JSU_PassRule(function(s) {
     if (s == null || s.length == 0) return 0;
     if (s.length < 5) return 3;
     if (s.length >= 5 && s.length < 8) return 6;
     if (s.length >= 8 && s.length < 16) return 12;
     if (s.length >= 16) return 18;
})
```

The code above creates a new rule that checks the length of a string (the password) and scores it accordingly. These values are the default used by the measuring algorithm, but you can create your own rules with your own values.

B. Properties

__func__

This is a private property, do not use directly. It is used to hold the actual rule function of the object.

C. Methods

1. getScore

Gets the score by executing a rule on a specified password. This method has no use outside the module, being used only by the JSU.PassMeter object to cumulate scores and get the strength of a password.

Syntax:

```
getScore(s)
```

Arguments:

- ***s***: the string to check (the password).

Returns: A number reflecting the score obtained by applying the rule.

D. Defaults

defaultRules

This is the default set of rules used by the JSU.PassMeter object to get the strength of a password.

JSU.PassMeter API Reference

A. Constructor

Syntax:

```
JSU.PassMeter([rules [, interpretation]])
```

Arguments:

- ***rules***: an array of JSU.PassRule objects.

Returns: Nothing.

B. Properties

1. __rules__

This is a private property, do not use it directly. If you want to change the measuring rules, use the **changeRules** mehod instead. It holds the array of rules to execute on a password.

2. __intpt__

This is a private property, do not use it directly. If you want to change the interpretation method, use the **changeInterpretation** method instead. It holds the actual function used to interpret the score obtained by the given password.

C. Methods

1. changeRules

This method is used to change the default rules of measuring the strength.

Syntax:

```
changeRules(newRules)
```

Arguments:

- ***rules***: an array of JSU.PassRule objects.

Returns: Nothing.

2. changeInterpretation

This method is used to change the default interpretation method for a given strength score.

Syntax:

```
changeInterpretation(newIntpt)
```

Arguments:

- ***intpt***: a function used to interpret the score.

Returns: Usually a string containing the interpretation of the strength score, but can return anything you need to correctly interpret the score.

3. getStrength

Gets the cumulative score obtained by applying all the rules to a password.

Syntax:

```
getStrength(s)
```

Arguments:

- ***s***: the password to measure.

Returns: A number

.

D. Defaults

defaultInterpretation

This is the default interpretation function used by the JSU.PassMeter object interpret the results of a password strength check.

The JSU.PassMeter in action

In this part of this book I'll show you how to use the JSU.PassMeter to get the strength of a password. Here's the example code:

```
var p = new JSU.PassMeter();
alert(p.getStrength("a%_2")); // good
alert(p.getStrength("a")); // very weak
alert(p.getStrength("Aa_2^/a01b02")); // very strong
```

That's pretty much it. But this example uses the default set of rules and the default interpretation function. Next, I'll show you how to create a new interpretation method based on the same default rules, but using your own strings (good for localization purposes).

This is the code you need to write in order to have a custom interpreting method:

```
var i = function(score) {
    if (score <= 11) return "no way I'm gonna let you in!";
    if (score > 11 && score <= 21) return "keep trying!";
     if (score > 21 && score <= 31) return "close, but not close enough!";
     if (score > 31 && score <= 41) return "are you sure ?";
     if (score > 41) return "you're in!";
};

var p = new JSU.PassMeter(JSU.PassRule.defaultRules, i);
alert(p.getStrength("a%_2")); // close, but not close enough!
alert(p.getStrength("a")); // no way I'm gonna let you in!
alert(p.getStrength("Aa_2^/a01b02")); // you're in!
```

Now let's explain the code. First, I've recreated the default interpretation method and changed the resulting string values. Then, I've created a new JSU.PassMeter object using the default rules and our previously defined interpretation method. The results are as expected. Notice that the default set of rules can be accessed via *JSU.PassRule.defaultRules*, while the default interpretation method is *JSU.PassMeter.defaultInterpretation*.

Creating a custom strength meter

Let's say we want to create a strength meter that checks if a password is good or bad, nothing complicated. We could create a new rule for checking the password's length, and another rule for checking if the password has at least one underscore. The password will be *good* if its length is greater than 5 and contains at least one underscore, otherwise it will be a *bad* password. Let's first look at the code:

```
var r = [
    new JSU.PassRule(function(s) {
        if (JSU.String.trim(s).length > 5) return 1;
        return 0;
    }),
    new JSU.PassRule(function(s) {
        if (s.indexOf("_") > -1) return 1;
        return 0;
    })
];
var i = function(score) {
    // if both rules are checked, the overall score is 2
    if (score == 2) return "good";
    return "bad"
}

var p = new JSU.PassMeter(r, i);
alert(p.getStrength("b_ad")); // bad
alert(p.getStrength("bad")); // bad
alert(p.getStrength("good_one")); // good
```

First I've created the set of rules, an array of JSU.PassRule objects that define the actual rules. The first rule checks if the password's length is greater than 5 character long and, if so, will return a score of 1 (meaning that it passed that rule), otherwise it will return 0 points (it is good to always end the rule by returning a 0, in case no condition is met and to avoid possible bugs or wrong condition spelling).

After the rules are set, I've created a new interpretation method which interprets the score obtained by the password. If both rules apply, the score will be 2, while if only one applies, the score will be 1. Obviously, if none applies, the score is 0. So, the interpretation method will check if the score is 2, meaning that the password is "*good*", otherwise it returns a "*bad*" result.