# Seahorse

## User guide



José Facundo Maldonado

# Seahorse
# User guide

by José Facundo Maldonado

# Index

# 1 - Introduction

Seahorse is a library of JavaScript, licensed as free software, created to simplify the use of forms, particularly the validation of them. Provides functions for validation, conversion and behavior for handling dates, times, numbers, text and e-mail addresses or URLs.

All the functions are highly configurable, allowing to specify the range of valid values, illegal characters, formats, and, in the case of behaviors, responses to the events of loss of focus and a key pressed.

It can be used with any JavaScript framework, however, has a plugin to be used along with jQuery.

## 2 - Validation

The validation functions are responsible for checking if a string represents or not a particular data type. In addition to text, the functions receive some values as parameters, that determine the criteria by which the validation is performed.

### 2.1 - isNumber()

The function `isNumber()` checks if a string represents a number. It receives as parameters the string, the character used as decimal separator and the character used as grouping separator (also known as the thousands separator).

```
Seahorse.isNumber("1,000,000.00", '.', ',')  ➡ true
Seahorse.isNumber("1 000 000.00", ',', '.')  ➡ false
Seahorse.isNumber("1 000 000.00", ',', ' ')  ➡ true
Seahorse.isNumber("1.000.000,00", ',', '.')  ➡ true
```

### 2.2 - isInteger()

The function `isInteger()` checks if a string represents an integer. It receives as parameters the string and the character used as grouping separator (also known as the thousands separator).

```
Seahorse.isInteger("1,000,000", ',')  ➡ true
Seahorse.isInteger("1 000 000", ',')  ➡ false
Seahorse.isInteger("1 000 000", ' ')  ➡ false
Seahorse.isInteger("1.000.000", '.')  ➡ true
```

### 2.3 - isNumeric()

The function `isNumeric()` checks if a string contains only numbers.

```
Seahorse.isNumeric("1234")  ➡ true
Seahorse.isNumeric("1,234")  ➡ false
```

### 2.4 - isAlphabetical()

The function `isAlphabetical()` checks if a string contains only letters.

```
Seahorse.isAlphabetical("abcdñ")  ➡ true
Seahorse.isAlphabetical("abcñ123")  ➡ false
```

### 2.5 - isAlphanumeric()

The function `isAlphanumeric()` checks if a string contains only letters and numbers.

```
Seahorse.isAlphanumeric("abcdñ123")  ➡ true
Seahorse.isAlphanumeric("abcñ-123")  ➡ false
```

### 2.6 - isAlphabeticalAscii()

The function `isAlphabeticalAscii()` checks if a string contains only letters of the basic latin alphabet.

```
Seahorse.isAlphabeticalAscii("abcd")  ➡ true
Seahorse.isAlphabeticalAscii("abcdñ")  ➡ false
```

## 2.7 - isAlphanumericAscii()

The function `isAlphanumericAscii()` checks if a string contains only numbers and letters of the basic latin alphabet.

```
Seahorse.isAlphanumericAscii("abc123")    ➡ true
Seahorse.isAlphanumericAscii("abcñ123")   ➡ false
```

## 2.8 - isAsciiText()

The function `isAsciiText()` checks if a string contains only characters of the ASCII character-encoding scheme.

```
Seahorse.isAsciiText("abcd (123) @#")     ➡ true
Seahorse.isAsciiText("abcd (123) @#ñá")   ➡ false
```

## 2.9 - isIPv4()

The function `isIPv4()` checks if a string represents an IP version 4 address.

```
Seahorse.isIPv4("192.168.0.1")    ➡ false
Seahorse.isIPv4("192.168.0.256")  ➡ false
```

## 2.10 - isIPv6()

The function `isIPv6()` checks if a string represents an IP version 6 address.

```
Seahorse.isIPv6("11:22:33:44:55:66:77:88")  ➡ true
Seahorse.isIPv6("11:22:33:44:55:::")        ➡ true
```

## 2.11 - isEmail()

The function `isEmail()` checks if a string represents an e-mail address.

```
Seahorse.isEmail("test@test.com")  ➡ true
Seahorse.isEmail("test@test")      ➡ false
```

## 2.12 - isHttp()

The function `isHttp()` checks if a string represents a HTTP address.

```
Seahorse.isHttp("http://www.test.com")  ➡ true
Seahorse.isHttp("www.test.com")         ➡ false
```

## 2.13 - isFtp()

The function `isFtp()` checks if a string represents a FTP address.

```
Seahorse.isFtp("ftp://ftp.test.com")  ➡ true
Seahorse.isFtp("ftp.test.com")        ➡ false
```

## 2.14 - isDate()

The function `isDate()` checks if a string represents a date. It receives, as parameters, the string and the format of the date, made by the combination of:

`d` - day of the month (no leading zero)
`dd` - day of the month (two digits)
`m` - month of year (no leading zero)
`mm` - month of year (two digits)
`yy` - year (two digits)
`yyyy` - year (four digits)

The format is used only to determine the order of the figures, and for that reason, is equivalent, for example, used as the format `dd/mm/yyyy` to `dd-mm-yyyy`.

```
Seahorse.isDate("31/07/2010","dd/mm/yyyy")  ➡ true
Seahorse.isDate("31-07-2010","dd/mm/yyyy")  ➡ true
Seahorse.isDate("07/31/2010","mm/dd/yyyy")  ➡ true
Seahorse.isDate("07 31 2010","mm/dd/yyyy")  ➡ true
Seahorse.isDate("07/32/2010","mm/dd/yyyy")  ➡ false
Seahorse.isDate("06/31/2010","mm/dd/yyyy")  ➡ false
```

## 2.15 - isTime()

The function `isTime()` checks if a string represents an instant of time. It receives, as parameters, the string and the format of the time, made by the combination of:

`s` - seconds (no leading zero)
`ss` - seconds (two digits)
`m` - minutes (no leading zero)
`mm` - minutes (two digits)
`h` - hours (two digits)
`hh` - hours (four digits)

The format is used only to determine the order of the figures, and for that reason, is equivalent, for example, used as the format `hh:mm:ss` to `hh-mm-ss`.

```
Seahorse.isTime("06:20:22","hh:mm:ss")  ➡ true
Seahorse.isTime("06-20-22","hh:mm:ss")  ➡ true
Seahorse.isTime("22:20:06","s:m:h")     ➡ true
Seahorse.isTime("22 20 06","s:m:h")     ➡ true
Seahorse.isTime("22:20:60","hh:mm:ss")  ➡ false
Seahorse.isTime("24:20:22","hh:mm:ss")  ➡ false
```

# 3 - Parsing

The parsing functions are responsible for analyzing a text string to convert into another type of data or give a particular format. In addition to text, the functions receive, as parameters, some values that determine the criteria by which the analysis and the conversion are performed.

## 3.1 - parseNumber()

The function `parseNumber()` parses a string and returns a number. It receives as parameters the string, the character used as decimal separator and the character used as grouping separator (also known as the thousands separator).

```
Seahorse.parseNumber("1,234.5", '.', ',')      ➡ 1234.5
Seahorse.parseNumber("1,234.5", ',', '.')      ➡ 1.2345
Seahorse.parseNumber("1234 5", ',', '.')       ➡ 1234
Seahorse.parseNumber("1234sdfg5", ',', '.')    ➡ 1234
```

## 3.2 - parseInteger()

The function `parseInteger()` parses a string and returns an integer. It receives as parameters the string and the character used as grouping separator (also known as the thousands separator).

```
Seahorse.parseInteger("1,234", ',')     ➡ 1234
Seahorse.parseInteger("1.234.5", '.')   ➡ 12345
Seahorse.parseInteger("1 234", ' ')     ➡ 1234
Seahorse.parseInteger("1 234", ',')     ➡ 1
```

## 3.3 - parseIPv4()

The function `parseIPv4()` parses a string that represents an IP version 4 address and returns a array with four numbers.

```
Seahorse.parseIPv4("192.168.0.1")      ➡ 192,168,0,1
Seahorse.parseIPv4("192.168.0.256")    ➡ null
Seahorse.parseIPv4("192 168 0 1")      ➡ null
```

## 3.4 - parseIPv6()

The function `parseIPv6()` parses a string that represents an IP version 6 address and returns a array with eight hexadecimal numbers.

```
Seahorse.parseIPv6("11:22:33:44:55:66:77:88")      ➡ 0x0011,0x0022,0x0033,0x0044,
                                                     0x0055,0x0066,0x0077,0x0088
Seahorse.parseIPv6("11:22:33:44:55:66:77:FFFG")    ➡ null
Seahorse.parseIPv6("11 22 33 44 55 66 77 88")      ➡ null
```

## 3.5 - parseDate()

The function `parseDate()` receives a string representing a date and transforms it according to a specific format. It receives, as parameters, the string, a boolean indicating if the empty fields has to be completed with the actual date and the format of the date, made by the combination of:

- `d` - day of the month (no leading zero)
- `dd` - day of the month (two digits)
- `m` - month of year (no leading zero)
- `mm` - month of year (two digits)

`yy` - year (two digits)
`yyyy` - year (four digits)

```
Seahorse.parseDate("31/07/2010","dd/mm/yyyy")      ➡ 31/07/2010
Seahorse.parseDate("31-07-2010","dd/mm/yyyy")      ➡ 31/07/2010
Seahorse.parseDate(" 07/31 (2010)","mm/dd/yyyy")   ➡ 07/31/2010
Seahorse.parseDate("07/32/2010","mm/dd/yyyy")      ➡ null
Seahorse.parseDate("07/31","mm/dd/yyyy")           ➡ null
Seahorse.parseDate("07/31","mm/dd/yyyy",true)      ➡ 07/31/2010
Seahorse.parseDate("07","mm/dd/yyyy",false)        ➡ null
Seahorse.parseDate("07","mm/dd/yyyy",true)         ➡ 07/20/2010
```

## 3.6 - parseTime()

The function `parseTime()` receives a string representing a instant of time and transforms it according to a specific format. It receives, as parameters, the string, a boolean indicating if the empty fields has to be completed with the actual hour and the format of the time, made by the combination of:

`s` - seconds (no leading zero)
`ss` - seconds (two digits)
`m` - minutes (no leading zero)
`mm` - minutes (two digits)
`h` - hours (two digits)
`hh` - hours (four digits)

```
Seahorse.parseTime("02:05:33","hh:mm:ss")      ➡ 02:05:33
Seahorse.parseTime("02-05-33","hh:mm:ss")      ➡ 02:05:33
Seahorse.parseTime(" 02/05 (33)","hh:mm:ss")   ➡ 02:05:33
Seahorse.parseTime("02:60:33","hh:mm:ss")      ➡ null
Seahorse.parseTime("02:05","hh:mm:ss")         ➡ null
Seahorse.parseTime("02:05","hh:mm:ss",true)    ➡ 02:05:12
Seahorse.parseTime("07","hh:mm:ss",false)      ➡ null
Seahorse.parseTime("07","hh:mm:ss",true)       ➡ 07:50:12
```

# 4 - Behavior

The behaviors allow real-time validation of form fields by events of loss of focus and a key pressed. The behaviors are assigned by different functions that receive, as parameter, the id or reference of the field, the options for validation and response options.

The validation options determines how the field is validated and are similar to the parameters received by the validation functions. The response options determines the actions to be undertaken when it detects that a field has a valid or invalid value.

## 4.1 - Texts

To define inputs that accept textual data, there are the methods `text()`, which by itself does not impose any restriction, `alphanumeric()`, which accepts only letters and numbers (does not allow symbols or punctuation), `alphabetical()`, which only accepts letters, and `numeric()`, which only accepts numbers.

As validation options, all the functions accept the parameters: `notEmpty`, `minLength`, `maxLength`, `asciiCharacters`, `requiredCharacters`, `forbiddenCharacters`, `allowedCharacters` and `aditionalValidation`.

`notEmpty` allows to indicate whether a field can be left blank or not. The default is false (fields can be left blank).

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      Seahorse.text("textNe1", {}, {"okClass": "ok", "errorClass": "error"});
      document.getElementById("textNe1").seahorse.verify();

      Seahorse.text("textNe2", {"notEmpty": false}, {"okClass": "ok", "errorClass":
"error"});
      document.getElementById("textNe2").seahorse.verify();

      Seahorse.text("textNe3", {"notEmpty": true}, {"okClass": "ok", "errorClass":
"error"});
      document.getElementById("textNe3").seahorse.verify();
      }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table>
      <tr>
         <td>text()</td>
         <td></td>
         <td><input type="text" id="textNe1" value=""/></td>
      </tr>
      <tr>
         <td>text()</td>
         <td>notEmpty = true</td>
         <td><input type="text" id="textNe2" value=""/></td>
      </tr>
      <tr>
         <td>text()</td>
         <td>notEmpty = false</td>
```

```
        <td><input type="text" id="textNe3" value=""/></td>
      </tr>
   </table></center>
   </form>

</body>
</html>
```

`minLength` and `maxLength` let you set the minimum and maximum number of characters that can have an input.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
     {
     Seahorse.text("textMl1", {"notEmpty": true, "minLength" : 3},
                {"okClass": "ok", "errorClass": "error"});
     document.getElementById("textMl1").seahorse.verify();

     Seahorse.text("textMl2", {"maxLength" : 6}, {"okClass": "ok", "errorClass":
"error"});
     document.getElementById("textMl2").seahorse.verify();
     }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
     <tr>
        <td>text()</td>
        <td>notEmpty = true, minLength = 3</td>
        <td><input type="text" id="textMl1" value=""/></td>
     </tr>
     <tr>
        <td>text()</td>
        <td>maxLengt = 6</td>
        <td><input type="text" id="textMl2" value=""/></td>
     </tr>
   </table></center>
   </form>

</body>
</html>
```

`asciiCharacters` allows to indicate whether the entry will only allow ASCII characters, rather than allow all Unicode characters. By default, all Unicode characters are allowed.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
```

```
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
     {
     Seahorse.text("textAc1", {}, {"okClass": "ok", "errorClass": "error"});
     document.getElementById("textAc1").seahorse.verify();

     Seahorse.text("textAc2", {"asciiCharacters": false}, {"okClass": "ok", "errorClass":
"error"});
     document.getElementById("textAc2").seahorse.verify();

     Seahorse.text("textAc3", {"asciiCharacters": true}, {"okClass": "ok", "errorClass":
"error"});
     document.getElementById("textAc3").seahorse.verify();
     }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo codigo">
     <tr>
        <td>text()</td>
        <td></td>
        <td><input type="text" id="textAc1" value=""/></td>
     </tr>
     <tr>
        <td>text()</td>
        <td>asciiCharacters = false</td>
        <td><input type="text" id="textAc2" value=""/></td>
     </tr>
     <tr>
        <td>text()</td>
        <td>asciiCharacters = true</td>
        <td><input type="text" id="textAc3" value=""/></td>
     </tr>
   </table></center>
   </form>

</body>
</html>
```

`requiredCharacters` lets you specify the characters that has to be entered for the field be considered valid.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
     {
     Seahorse.text("textRc1", {"requiredCharacters": "_"},
          {"okClass": "ok", "errorClass": "error"});
     document.getElementById("textRc1").seahorse.verify();

     Seahorse.alphanumeric("textRc2", {"requiredCharacters": "a1"},
          {"okClass": "ok", "errorClass": "error"});
```

```
     document.getElementById("textRc2").seahorse.verify();

     Seahorse.alphabetical("textRc3", {"requiredCharacters": ["a".charCodeAt(0),
"b".charCodeAt(0)]},
          {"okClass": "ok", "errorClass": "error"});
     document.getElementById("textRc3").seahorse.verify();
     }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo codigo">
     <tr>
        <td>text()</td>
        <td>requiredCharacters = "_"</td>
        <td><input type="text" id="textRc1" value=""/></td>
     </tr>
     <tr>
        <td>alphanumeric()</td>
        <td>requiredCharacters = "a1"</td>
        <td><input type="text" id="textRc2" value=""/></td>
     </tr>
     <tr>
        <td>alphabetical()</td>
        <td>requiredCharacters = ["a", "b"]</td>
        <td><input type="text" id="textRc3" value=""/></td>
     </tr>
   </table></center>
   </form>

</body>
</html>
```

`forbiddenCharacters` allows to indicate the characters that must not be entered in order that the entry be considered to be valid.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
     {
     Seahorse.text("textFc1", {"forbiddenCharacters": "_"},
             {"okClass": "ok", "errorClass": "error"});
     document.getElementById("textFc1").seahorse.verify();

     Seahorse.alphanumeric("textFc2", {"forbiddenCharacters": "a1"},
             {"okClass": "ok", "errorClass": "error"});
     document.getElementById("textFc2").seahorse.verify();

     Seahorse.alphabetical("textFc3", {"forbiddenCharacters": ["a".charCodeAt(0),
"b".charCodeAt(0)]},
             {"okClass": "ok", "errorClass": "error"});
     document.getElementById("textFc3").seahorse.verify();
     }
</script>
</head>
```

```
<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo codigo">
     <tr>
        <td>text()</td>
        <td>forbiddenCharacters = "_"</td>
        <td><input type="text" id="textFc1" value=""/></td>
     </tr>
     <tr>
        <td>alphanumeric()</td>
        <td>forbiddenCharacters = "a1"</td>
        <td><input type="text" id="textFc2" value=""/></td>
     </tr>
     <tr>
        <td>alphabetical()</td>
        <td>forbiddenCharacters = ["a", "b"]</td>
        <td><input type="text" id="textFc3" value=""/></td>
     </tr>
   </table></center>
   </form>

</body>
</html>
```

`allowedCharacters` allows to indicate a group of characters that an entry can have and nonetheless be considered valid. This parameter allows, for example, that an entry of type alphabetical can accepts some signs of punctuation.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
     {
     Seahorse.alphanumeric("textAllc1", {"allowedCharacters": " "},
             {"okClass": "ok", "errorClass": "error"});
     document.getElementById("textAllc1").seahorse.verify();

     Seahorse.alphabetical("textAllc2", {"allowedCharacters": "123"},
             {"okClass": "ok", "errorClass": "error"});
     document.getElementById("textAllc2").seahorse.verify();

     Seahorse.numeric("textAllc3", {"allowedCharacters": ["a".charCodeAt(0),
"b".charCodeAt(0)]},
             {"okClass": "ok", "errorClass": "error"});
     document.getElementById("textAllc3").seahorse.verify();
     }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo codigo">
     <tr>
        <td>alphanumeric()</td>
        <td>allowedCharacters = " "</td>
        <td><input type="text" id="textAllc1" value=""/></td>
     </tr>
```

```
      <tr>
         <td>alphabetical()</td>
         <td>allowedCharacters = "123"</td>
         <td><input type="text" id="textAllc2" value=""/></td>
      </tr>
      <tr>
         <td>numeric()</td>
         <td>allowedCharacters = ["a", "b"]</td>
         <td><input type="text" id="textAllc3" value=""/></td>
      </tr>
   </table></center>
   </form>

</body>
</html>
```

`aditionalValidation` allows specify a function to perform an additional verification to determine if the input is valid or not.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      Seahorse.text("textAv1", {"aditionalValidation": oddLength},
         {"okClass": "ok", "errorClass": "error"});
      document.getElementById("textAv1").seahorse.verify();

      Seahorse.text("textAv2", {"aditionalValidation": evenLength},
         {"okClass": "ok", "errorClass": "error"});
      document.getElementById("textAv2").seahorse.verify();
      }

   function oddLength(element)
      { return element.value.length%2 == 1;}

   function evenLength(element)
      { return element.value.length%2 == 0;}
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo codigo">
      <tr>
         <td>text()</td>
         <td>aditionalValidation = oddLength()</td>
         <td><input type="text" id="textAv1" value=""/></td>
      </tr>
      <tr>
         <td>text()</td>
         <td>aditionalValidation = evenLenght()</td>
         <td><input type="text" id="textAv2" value=""/></td>
      </tr>
   </table></center>
   </form>

</body>
```

```
</html>
```

## 4.2 - Numbers

To set numeric inputs you should use the functions `number()`, for numbers of all types, and `integer()`, for integers.

As validation options, the functions accept the parameters: `groupingCharacter`, `decimalCharacter`, `minValue`, `maxValue`, `notEmpty` and `aditionalValidation`.

`decimalCharacter` and `groupingCharacter` are used to define which characters are used, respectively, as decimal separator and grouping character (or thousands separator).

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
     {
     Seahorse.number( "number21", {"decimalCharacter": '.', "groupingCharacter" : ','},
             {"okClass": "ok", "errorClass": "error"} );
     document.getElementById("number21").seahorse.verify();

     Seahorse.number( "number22", {"decimalCharacter": ',', "groupingCharacter" : '.'},
             {"okClass": "ok", "errorClass": "error"} );
     document.getElementById("number22").seahorse.verify();

     Seahorse.number( "number23", {"decimalCharacter": '.', "groupingCharacter" : ' '},
             {"okClass": "ok", "errorClass": "error"} );
     document.getElementById("number23").seahorse.verify();
     }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
     <tr>
        <td>number()</td>
        <td>decimalCharacter = '.', groupingCharacter = ','</td>
        <td><input type="text" id="number21" value=""/></td>
     </tr>
     <tr>
        <td>number()</td>
        <td>decimalCharacter = ',', groupingCharacter = '.'</td>
        <td><input type="text" id="number22" value=""/></td>
     </tr>
     <tr>
        <td>number()</td>
        <td>decimalCharacter = '.', groupingCharacter = ' '</td>
        <td><input type="text" id="number23" value=""/></td>
     </tr>
   </table></center>
   </form>

</body>
</html>
```

`minValue` and `maxValue` defines the minimum and maximum values that an input can have.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
     {
     Seahorse.number( "numberMv1",
        {"decimalCharacter": '.', "groupingCharacter" : ',',
        "minValue" : -10.5, "maxValue" : 10.5},
        {"okClass": "ok", "errorClass": "error"} );
     document.getElementById("numberMv1").seahorse.verify();

     Seahorse.number( "numberMv2",
        {"decimalCharacter": '.', "groupingCharacter" : ',',
        "minValue" : 0, "maxValue" : 10.5},
        {"okClass": "ok", "errorClass": "error"} );
     document.getElementById("numberMv2").seahorse.verify();

     Seahorse.number( "numberMv3",
        {"decimalCharacter": '.', "groupingCharacter" : ',',
        "minValue" : -10.5, "maxValue" : 0},
        {"okClass": "ok", "errorClass": "error"} );
     document.getElementById("numberMv3").seahorse.verify();

     Seahorse.integer( "numberMv4",
        {"groupingCharacter" : ',', "minValue" : -10, "maxValue" : 10},
        {"okClass": "ok", "errorClass": "error"} );
     document.getElementById("numberMv4").seahorse.verify();

     Seahorse.integer( "numberMv5",
        {"groupingCharacter" : ',', "minValue" : 0, "maxValue" : 10},
        {"okClass": "ok", "errorClass": "error"} );
     document.getElementById("numberMv5").seahorse.verify();

     Seahorse.integer( "numberMv6",
        {"groupingCharacter" : ',', "minValue" : -10, "maxValue" : 0},
        {"okClass": "ok", "errorClass": "error"} );
     document.getElementById("numberMv6").seahorse.verify();
     }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
     <tr>
        <td>number()</td>
        <td>
          decimalCharacter = '.', groupingCharacter = ','<br/>
          minValue = -10.5, maxValue = 10.5
        </td>
        <td><input type="text" id="numberMv1" value=""/></td>
     </tr>
     <tr>
        <td>number()</td>
        <td>
          decimalCharacter = '.', groupingCharacter = ','<br/>
          minValue = 0, maxValue = 10.5
```

```
        </td>
        <td><input type="text" id="numberMv2" value=""/></td>
      </tr>
      <tr>
        <td>number()</td>
        <td>
          decimalCharacter = '.', groupingCharacter = ','<br/>
          minValue = -10.5, maxValue = 0
        </td>
        <td><input type="text" id="numberMv3" value=""/></td>
      </tr>
      <tr>
        <td>integer()</td>
        <td>
          groupingCharacter = ','<br/>
          minValue = -10, maxValue = 10
        </td>
        <td><input type="text" id="numberMv4" value=""/></td>
      </tr>
      <tr>
        <td>integer()</td>
        <td>
          groupingCharacter = ','<br/>
          minValue = 0, maxValue = 10
        </td>
        <td><input type="text" id="numberMv5" value=""/></td>
      </tr>
      <tr>
        <td>integer()</td>
        <td>
          groupingCharacter = ','<br/>
          minValue = -10, maxValue = 0
        </td>
        <td><input type="text" id="numberMv6" value=""/></td>
      </tr>
    </table></center>
    </form>

</body>
</html>
```

`notEmpty` and `aditionalValidation` work in the same way as in the case of texts.

## 4.3 - Dates and hours

To set date and time inputs you should use the functions `date()`, for dates, and `time()`, for hours.

As validation options, the functions accept the parameters: `format`, `autofill`, `minValue`, `maxValue`, `notEmpty` and `aditionalValidation`.

`format` defines the format of the date or time. The format for dates is constituted by the combination of 'd' or 'dd' (day one or two digits), 'm' or 'mm' (month of one or two digits) and 'y' or 'yyyy' (years two to four digits). The format for times s constituted by the combination of 's' or 'ss' (seconds of one or two digits), 'm' or 'mm' (minutes of one or two digits) and 'h' or 'hh' (hours of one or two digits).

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
```

```
     {
     Seahorse.date( "date21", {"format" : 'mm/dd/yyyy'}, {"okClass": "ok", "errorClass":
"error"} );
     document.getElementById("date21").seahorse.verify();

     Seahorse.date( "date22", {"format" : 'dd/mm/yyyy'}, {"okClass": "ok", "errorClass":
"error"} );
     document.getElementById("date22").seahorse.verify();

     Seahorse.date( "date23", {"format" : 'yyyy-mm-dd'}, {"okClass": "ok", "errorClass":
"error"} );
     document.getElementById("date23").seahorse.verify();

     Seahorse.time( "time21", {"format" : 'hh:mm:ss'}, {"okClass": "ok", "errorClass":
"error"} );
     document.getElementById("time21").seahorse.verify();

     Seahorse.time( "time22", {"format" : 'hh-mm-ss'}, {"okClass": "ok", "errorClass":
"error"} );
     document.getElementById("time22").seahorse.verify();
     }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
      <tr>
         <td>date()</td>
         <td>format = mm/dd/yyyy</td>
         <td><input type="text" id="date21" value=""/></td>
      </tr>
      <tr>
         <td>date()</td>
         <td>format = dd/mm/yyyy</td>
         <td><input type="text" id="date22" value=""/></td>
      </tr>
      <tr>
         <td>date()</td>
         <td>format = yyyy-mm-dd</td>
         <td><input type="text" id="date23" value=""/></td>
      </tr>
      <tr>
         <td>time()</td>
         <td>format = hh:mm:ss</td>
         <td><input type="text" id="time21" value=""/></td>
      </tr>
      <tr>
         <td>time()</td>
         <td>format = hh-mm-ss</td>
         <td><input type="text" id="time22" value=""/></td>
      </tr>
   </table></center>
   </form>

</body>
</html>
```

`autofill` indicate if the incompleted fields of a date or a time must be completed with the actual date or time.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
```

```
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
     {
     Seahorse.date( "date31", {"format" : 'yyyy-mm-dd'},
        {"okClass": "ok", "errorClass": "error"} );
     document.getElementById("date31").seahorse.verify();

     Seahorse.date( "date32", {"format" : 'yyyy-mm-dd', 'autofill': true},
        {"okClass": "ok", "errorClass": "error"} );
     document.getElementById("date32").seahorse.verify();

     Seahorse.date( "date33", {"format" : 'yyyy-mm-dd', 'autofill': false},
        {"okClass": "ok", "errorClass": "error"} );
     document.getElementById("date33").seahorse.verify();

     Seahorse.time( "time31", {"format" : 'hh:mm:ss'},
        {"okClass": "ok", "errorClass": "error"} );
     document.getElementById("time31").seahorse.verify();

     Seahorse.time( "time32", {"format" : 'hh:mm:ss', 'autofill': true},
        {"okClass": "ok", "errorClass": "error"} );
     document.getElementById("time32").seahorse.verify();

     Seahorse.time( "time33", {"format" : 'hh:mm:ss', 'autofill': false},
{"okClass": "ok", "errorClass": "error"} );
     document.getElementById("time33").seahorse.verify();
     }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
     <tr>
        <td>date()</td>
        <td>format = yyyy-mm-dd</td>
        <td><input type="text" id="date31" value=""/></td>
     </tr>
     <tr>
        <td>date()</td>
        <td>format = yyyy-mm-dd, autofill = true</td>
        <td><input type="text" id="date32" value=""/></td>
     </tr>
     <tr>
        <td>date()</td>
        <td>format = yyyy-mm-dd, autofill = false</td>
        <td><input type="text" id="date33" value=""/></td>
     </tr>
     <tr>
        <td>time()</td>
        <td>format = hh:mm:ss</td>
        <td><input type="text" id="time31" value=""/></td>
     </tr>
     <tr>
        <td>time()</td>
        <td>format = hh:mm:ss, autofill = true</td>
        <td><input type="text" id="time32" value=""/></td>
     </tr>
     <tr>
        <td>time()</td>
        <td>format = hh:mm:ss, autofill = false</td>
        <td><input type="text" id="time33" value=""/></td>
```

```
      </tr>
   </table></center>
   </form>

</body>
</html>
```

`minValue` and `maxValue` defines the minimum and maximum values that can have an input.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
     {
     Seahorse.date( "date41", {"format" : 'dd/mm/yyyy',
        "minValue" : "01/01/1996", "maxValue" : "07/07/2010"},
        {"okClass": "ok", "errorClass": "error"} );
     document.getElementById("date41").seahorse.verify();

     Seahorse.date( "date42", {"format" : 'dd/mm/yyyy',
        "minValue" : "01/01/2010", "maxValue" : "31/12/2010"},
        {"okClass": "ok", "errorClass": "error"} );
     document.getElementById("date42").seahorse.verify();

     Seahorse.time( "time41", {"format" : 'hh:mm:ss',
        "minValue" : "00:00:00", "maxValue" : "12:00:00"},
        {"okClass": "ok", "errorClass": "error"} );
     document.getElementById("time41").seahorse.verify();

     Seahorse.time( "time42", {"format" : 'hh:mm:ss',
        "minValue" : "12:34:56", "maxValue" : "23:59:59"},
        {"okClass": "ok", "errorClass": "error"} );
     document.getElementById("time42").seahorse.verify();
     }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
     <tr>
        <td>date()</td>
        <td>format = dd/mm/yyyy, minValue = "01/01/1996", maxValue = "07/07/2010"</td>
        <td><input type="text" id="date41" value=""/></td>
     </tr>
     <tr>
        <td>date()</td>
        <td>format = dd/mm/yyyy, minValue = "01/01/2010", maxValue = "31/12/2010"</td>
        <td><input type="text" id="date42" value=""/></td>
     </tr>
     <tr>
        <td>time()</td>
        <td>format = hh:mm:ss, minValue = "00:00:00", maxValue = "12:00:00"</td>
        <td><input type="text" id="time41" value=""/></td>
     </tr>
     <tr>
        <td>time()</td>
        <td>format = hh:mm:ss, minValue = "12:34:56", maxValue = "23:59:59"</td>
```

```
        <td><input type="text" id="time42" value=""/></td>
      </tr>
   </table></center>
   </form>

</body>
</html>
```

`notEmpty` and `aditionalValidation` work in the same way as in the case of texts.

## 4.4 - Internet's addresses

The functions `ipAddress()`, `email()`, `http()` and `ftp()` allow, respectively, define fields for IP addresses, e-mail and HTTP or FTP address.

As validation options, the functions accept only `notEmpty` and `aditionalValidation` parameters, except `ipAddress()` that also accept the version parameter, that can take the values 4 or 6.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      Seahorse.ipAddress( "ipv42", {"version" : 4}, {"okClass": "ok", "errorClass":
"error"} );
      document.getElementById("ipv42").seahorse.verify();

      Seahorse.ipAddress( "ipv62", {"version" : 6}, {"okClass": "ok", "errorClass":
"error"} );
      document.getElementById("ipv62").seahorse.verify();

      Seahorse.email( "email2", {}, {"okClass": "ok", "errorClass": "error"} );
      document.getElementById("email2").seahorse.verify();

      Seahorse.http( "http2", {}, {"okClass": "ok", "errorClass": "error"} );
      document.getElementById("http2").seahorse.verify();

      Seahorse.ftp( "ftp2", {}, {"okClass": "ok", "errorClass": "error"} );
      document.getElementById("ftp2").seahorse.verify();
      }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
      <tr>
         <td>ipAddress()</td>
         <td>version = 4</td>
         <td><input type="text" id="ipv42" value=""/></td>
      </tr>
      <tr>
         <td>ipAddress()</td>
         <td>version = 6</td>
         <td><input type="text" id="ipv62" value=""/></td>
      </tr>
      <tr>
         <td>email()</td>
```

```
        <td></td>
        <td><input type="text" id="email2" value=""/></td>
      </tr>
      <tr>
        <td>http()</td>
        <td></td>
        <td><input type="text" id="http2" value=""/></td>
      </tr>
      <tr>
        <td>ftp()</td>
        <td></td>
        <td><input type="text" id="ftp2" value=""/></td>
      </tr>
   </table></center>
   </form>

</body>
</html>
```

## 4.5 - Response options

The response options are used to define the actions to perform when it's detected that a field has a valid or invalid value. Unlike validation options, its parameters are common to all the functions of behaviors: `okClass`, `errorClass`, `targetErrorClass`, `targetOkClass`, `targetId`, `hiddenElementId`, `callbackFunction`, `forbidEntrance`, `autoparse` and `errorMessage`.

The parameters `okClass` and `errorClass` defines the CSS classes added to the field when it has a valid or invalid value.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .colorGreen { color:green; }
   .colorRed { color:red; }
   .colorBlue { color:blue; }
   .colorYellow { color:yellow; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
     {
     Seahorse.alphabetical( "resOpts1a", {}, {"okClass": "colorGreen", "errorClass":
"colorRed",
        "forbidEntrance": false} );
     document.getElementById("resOpts1a").seahorse.verify();

     Seahorse.alphabetical( "resOpts1b", {}, {"okClass": "colorBlue", "errorClass":
"colorYellow",
        "forbidEntrance": false} );
     document.getElementById("resOpts1b").seahorse.verify();
     }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
     <tr>
        <td>alphabetical()</td>
        <td>okClass = 'colorGreen', errorClass = 'colorRed'</td>
        <td><input type="text" id="resOpts1a" value=""/></td>
     </tr>
     <tr>
```

```
        <td>alphabetical()</td>
        <td>okClass = 'colorBlue', errorClass = 'colorYellow'</td>
        <td><input type="text" id="resOpts1b" value=""/></td>
      </tr>
   </table></center>
   </form>

</body>
</html>
```

The parameters `targetErrorClass` and `targetOkClass` are used to determine the CSS classes added, after the field loses focus, to an HTML element when the field value is valid or invalid. The target parameter indicates the id of the element you add those classes.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .colorGreen { color:green; }
   .colorRed { color:red; }
   .colorBlue { color:blue; }
   .colorYellow { color:yellow; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
     {
     Seahorse.alphabetical( "resOpts2a", {},
        {"targetOkClass": "colorGreen", "targetErrorClass": "colorRed",
        "forbidEntrance": false, "targetId": "resOpts-targetA"} );

     Seahorse.alphabetical( "resOpts2b", {},
        {"targetOkClass": "colorBlue", "targetErrorClass": "colorYellow",
        "forbidEntrance": false, "targetId": "resOpts-targetB"} );
     }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
     <tr>
        <td>alphabetical()</td>
        <td>
          targetOkClass = 'colorGreen', targetErrorClass = 'colorRed',<br/>
          targetId = 'resOpts-targetA'
        </td>
        <td><input type="text" id="resOpts2a" value=""/></td>
        <td id="resOpts-targetA">target</td>
     </tr>
     <tr>
        <td>alphabetical()</td>
        <td>
          targetOkClass = 'colorBlue', targetErrorClass = 'colorYellow',<br/>
          targetId = 'resOpts-targetB'
        </td>
        <td><input type="text" id="resOpts2b" value=""/></td>
        <td id="resOpts-targetB">target</td>
     </tr>
   </table></center>
   </form>

</body>
</html>
```

The `hiddenElementId` parameter indicates the id of an HTML element that will hide or show depending if the value of the field is valid or invalid.

```
<html>

<head>
<title>Seahorse example</title>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
     {
     Seahorse.alphabetical( "resOpts3a", {},
        {"hiddenElementId": "resOpts-hiddenTargetA", "forbidEntrance": false} );
     document.getElementById("resOpts3a").seahorse.verify();

     Seahorse.alphabetical( "resOpts3b", {},
        {"hiddenElementId": "resOpts-hiddenTargetB", "forbidEntrance": false} );
     document.getElementById("resOpts3b").seahorse.verify();
     }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
     <tr>
        <td>alphabetical()</td>
        <td>hiddenElementId = 'resOpts-hiddenTargetA'</td>
        <td><input type="text" id="resOpts3a" value=""/></td>
        <td><span id="resOpts-hiddenTargetA">target</span></td>
     </tr>
     <tr>
        <td>alphabetical()</td>
        <td>hiddenElementId = 'resOpts-hiddenTargetB'</td>
        <td><input type="text" id="resOpts3b" value=""/></td>
        <td><span id="resOpts-hiddenTargetB">target</span></td>
     </tr>
   </table></center>
   </form>

</body>
</html>
```

The `callbackFunction` parameter lets you define a function that is invoked when the field loses focus and which tells you whether it has a valid or invalid value.

```
<html>

<head>
<title>Seahorse example</title>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
     {
     Seahorse.alphabetical( "resOpts4a", {},
        {"callbackFunction": function(elem, valid) {alert(valid);}, "forbidEntrance":
false} );

     Seahorse.numeric( "resOpts4b", {},
        {"callbackFunction": function(elem, valid) {alert(valid);}, "forbidEntrance":
false} );
     }
</script>
```

```
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
     <tr>
        <td>alphabetical()</td>
        <td>callbackFunction = function(elem, valid) {alert(valid);}</td>
        <td><input type="text" id="resOpts4a" value=""/></td>
     </tr>
     <tr>
        <td>numeric()</td>
        <td>callbackFunction = function(elem, valid) {alert(valid);}</td>
        <td><input type="text" id="resOpts4b" value=""/></td>
     </tr>
   </table></center>
   </form>

</body>
</html>
```

The `forbidEntrance` parameter lets you specify whether to preventing, through `onKeyPress` event, the entry of characters that would make invalid the value of the field. By default, it prevents the entry of such characters.

```
<html>

<head>
<title>Seahorse example</title>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
     {
     Seahorse.numeric( "resOpts5a", {}, {"forbidEntrance": false} );

     Seahorse.numeric( "resOpts5b", {}, {"forbidEntrance": true} );
     }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
     <tr>
        <td>numeric()</td>
        <td>forbidEntrance = 'false'</td>
        <td><input type="text" id="resOpts5a" value=""/></td>
     </tr>
     <tr>
        <td>numeric()</td>
        <td>forbidEntrance = 'true'</td>
        <td><input type="text" id="resOpts5b" value=""/></td>
     </tr>
   </table></center>
   </form>

</body>
</html>
```

The `autparse` parameter lets you specify whether to convert the contents of the field, after it loses focus of it, to avoid the appearance of unnecessary characters.

```
<html>
```

```
<head>
<title>Seahorse example</title>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
     {
     Seahorse.integer( "resOpts6a", {"groupingCharacter" : ','}, {"autoparse": false} );

     Seahorse.integer( "resOpts6b", {"groupingCharacter" : ','}, {"autoparse": true} );
     }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET">
   <center><table class="cuadro_ejemplo">
     <tr>
        <td>integer()</td>
        <td>groupingCharacter : ',', autparse = 'false'</td>
        <td><input type="text" id="resOpts6a" value=""/></td>
     </tr>
     <tr>
        <td>integer()</td>
        <td>groupingCharacter : ',', autparse = 'true'</td>
        <td><input type="text" id="resOpts6b" value=""/></td>
     </tr>
   </table></center>
   </form>

</body>
</html>
```

The `errorMessage` parameter defines a message that explains the conditions on which a value entered into the field to be considered valid. This parameter only makes sense to define when the form, of the input, has a behavior assigned through one of the functions of this library.

## 4.6 - Forms

The function `form()` allows to add a behavior to forms, whereby, before sending the form, checks if all fields have valid values. This function takes as parameters the id of the form, the function that is invoked when you try to send the form, but one or more of its fields have invalid values, and an error message.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
     {
     Seahorse.text("form0_username", {"notEmpty": true},
         {"okClass": "ok", "errorClass": "error", "errorMessage": "Username required"});
     document.getElementById("form0_username").seahorse.verify();

     Seahorse.email("form0_email", {"notEmpty": true},
         {"okClass": "ok", "errorClass": "error", "errorMessage": "E-mail address
required"});
     document.getElementById("form0_email").seahorse.verify();
```

```
     Seahorse.form("form0", function(msjs, array) {alert(msjs);}, "Submit error" );
     }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET" id="form0">
   <center><table class="cuadro_formulario">
     <tr>
        <td align="right">Username</td>
        <td align="left"><input id="form0_username" type="text" name="username" value=""/
></td>
     </tr>
     <tr>
        <td align="right">Password</td>
        <td align="left"><input type="password" name="password" value="1234"/></td>
     </tr>
     <tr>
        <td align="right" valign="top">Gender</td>
        <td align="left">
          <input type="radio" name="gender" value="M" checked/><br/>
          <input type="radio" name="gender" value="F"/>
        </td>
     </tr>
     <tr>
        <td align="right">E-mail</td>
        <td align="left"><input id="form0_email" type="text" name="email" value=""/></td>
     </tr>
     <tr>
        <td align="right">Receive news</td>
        <td align="left"><input type="checkbox" name="news" value="true"/><br/></td>
     </tr>
     <tr>
        <td align="right" valign="top">Preferences</td>
        <td align="left"><select name="preferences" multiple="multiple">
          <option>Option A</option>
          <option>Option B</option>
            <option>Option C</option>
            <option>Option D</option>
        </select></td>
     </tr>
     <tr>
        <td colspan="2" align="right">
          <button id="form_0_btn">Send</button>
        </td>
     </tr>
   </table></center>
   </form>

</body>
</html>
```

## 4.7 - The 'seahorse' object

When a behavior is assigned to an element, an object is created and stored as an attribute in the element. This object stores the validation and response options of the behavior of the element, as well as the definition of the functions `validate()`, `parse()` and `verify()`.

`validate()` and `verify()` verify whether the field has a valid value or not, with the difference that `verify()` also performs the actions defined by the response options, so sometimes it may be appropriate to invoke this function after the page loads.

`parse()`, if the field is a number, an integer, a date, an hour or an IP address, it returns the converted value, depending on type of field.

Since Seahorse behaviors make use of the onkeyup event, onBlur and onKeyPress to implement the real-time validation, on the object 'seahorse' the functions related to this events, and assigned before the

assignation of the behavior, are also stored. These functions are invoked after the functions relating to the real-time validation.

# 5 - Others functions

## 5.1 - Comparisons

The functions `compareDate()` and `compareTime()` takes two dates or two times, respectively, and check which one is bigger than the other. They receive as parameters the dates or times, his format, and returns a negative value if the first date or time is less than the second, zero if equal and positive value if the first date or time is greater than the second.

## 5.2 - Serialization

The function `serialize()` encodes all the values of the elements of a form as a string, using URL or JSON enconding.

## 6 - jQuery

With the Seahorse plugin for jQuery you can use the jQuery selectors to assign behaviors or see if the fields have valid values, reducing the number of lines of code and making easier the maintenance of the code.

Seahorse provides the functions `seaBehavior()`, which assigns behaviors to the selected fields, `seaForm()`, which assigns behaviors to the selected forms, `seaValidate()`, which verifies that all elements have valid values, and `seaVerify()` that invokes the method `verify()` of all the selected elements.

```
<html>

<head>
<title>Seahorse example</title>

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/jquery-1.4.2.min.js"></script>
<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript" src="js/seahorse.jquery-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      jQuery(".integer").seaBehavior( "integer",
         {"groupingCharacter" : ','},
         {"okClass": "ok", "errorClass": "error", "errorMessage": "Integer error"} );

      jQuery(".number").seaBehavior( "number",
         {"decimalCharacter": '.', "groupingCharacter" : ','},
         {"okClass": "ok", "errorClass": "error", "errorMessage": "Number error"} );

      jQuery(".date").seaBehavior( "date",
         {"format" : 'dd/mm/yyyy'},
         {"okClass": "ok", "errorClass": "error", "errorMessage": "Date error"} );

      jQuery(".time").seaBehavior( "time",
         {"format" : 'hh-mm-ss'},
         {"okClass": "ok", "errorClass": "error", "errorMessage": "Time error"} );

      jQuery(".alphabetical").seaBehavior( "alphabetical", {},
         {"okClass": "ok", "errorClass": "error", "errorMessage": "Alphabetical error"} );

      jQuery(".alphanumeric").seaBehavior( "alphanumeric", {},
         {"okClass": "ok", "errorClass": "error", "errorMessage": "Alphanumeric error"} );

      jQuery(".numeric").seaBehavior( "numeric", {},
         {"okClass": "ok", "errorClass": "error", "errorMessage": "Numeric error"} );

      jQuery(":input").seaVerify();
      jQuery("#formulario").seaForm(function(msjs, array) {alert(msjs);}, "Submit
error");
      }
</script>
</head>

<body onload="javascript:init();">

   <form action="" method="GET" id="formulario">
   <center><table class="cuadro_ejemplo">
      <tr>
         <td>number()</td>
         <td><input type="text" class="number" value="1,000.00"/></td>
      </tr>
      <tr>
         <td>integer()</td>
```

```
        <td><input type="text" class="integer" value="1,000"/></td>
      </tr>
      <tr>
        <td>date()</td>
        <td><input type="text" class="date" value="31/12/2000"/></td>
      </tr>
      <tr>
        <td>time()</td>
        <td><input type="text" class="time" value="23:59:59"/></td>
      </tr>
      <tr>
        <td>alphanumeric()</td>
        <td><input type="text" class="alphanumeric" value="Alphanumeric123"/></td>
      </tr>
      <tr>
        <td>alphabetical()</td>
        <td><input type="text" class="alphabetical" value="Alphabetical"/></td>
      </tr>
      <tr>
        <td>numeric()</td>
        <td><input type="text" class="numeric" value="1234"/></td>
      </tr>
      <tr>
        <td colspan="2" align="right">
           <input type="submit" value="Submit"/>
        </td>
      </tr>
   </table></center>
   </form>

</body>
</html>
```

# 7 - Annex

## 7.1 - API specification

### 7.1.1 - Sumary

**About Seahorse**

| | |
|---|---|
| *Introduction* | Seahorse is an open source JavaScript library created for simplify the use of forms, particularly the form validation. |
| *License* | This library is licensed under the LGPL Version 3 license. |

**Validation behaviors**

| | |
|---|---|
| *Validation options* | The validation options are parameters that restrict the values, or the format, that the data of an input can have to be considered valid. |
| *Response options* | The response options are parameters that determines the courses of actions when an user enter a valid or invalid input. |

Functions

*Validation behaviors*

| | |
|---|---|
| *text()* | Gives to an element a text input behavior. |
| *alphabetical()* | Gives to an element an alphabetical input behavior. |
| *alphanumeric()* | Gives to an element an alphanumeric input behavior. |
| *numeric()* | Gives to an element a numeric input behavior. |
| *number()* | Gives to an element a float input behavior. |
| *integer()* | Gives to an element an integer input behavior. |
| *date()* | Gives to an element a date input behavior. |
| *time()* | Gives to an element a time input behavior. |
| *ipAddress()* | Gives to an element an IP address input behavior. |
| *email()* | Gives to an element an e-mail address input behavior. |
| *http()* | Gives to an element a HTTP address input behavior. |
| *ftp()* | Gives to an element a FTP address input behavior. |
| *form()* | Gives to an element a form behavior. |
| *removeBehavior()* | Removes, from a element, any behavior asigned by a function of the Seahorse's library. |

***Validation functions***

| | |
|---|---|
| *isNumber()* | Checks if a string represents a number. |
| *isInteger()* | Checks if a string represents an integer. |
| *isNumeric()* | Checks if a string contains only numbers. |
| *isAlphabetical()* | Checks if a string contains only alphabetical characters (including accents and another not Ascii characters). |
| *isAlphanumeric()* | Checks if a string contains only alphanumeric characters. |
| *isAlphabeticalAscii()* | Checks if a string contains only alphabetical ASCII characters. |
| *isAlphanumericAscii()* | Checks if a string contains only alphanumeric ASCII characters. |
| *isAsciiText()* | Checks if a string contains only ASCII characters. |
| *isIPv4()* | Checks if a string represents an IP version 4 address. |
| *isIPv6()* | Checks if a string represents an IP version 6 address. |
| *isEmail()* | Checks if a string represents an e-mail address. |
| *isHttp()* | Checks if a string represents a HTTP address. |
| *isFtp()* | Checks if a string represents a FTP address. |
| *isMonth()* | Checks if a string represents a month. |
| *isDay()* | Checks if a string represents a day of a particular month. |
| *isDate()* | Checks if string represents a date. |
| *isTime()* | Checks if string represents an instant of time. |
| *isDateFormat()* | Checks if string is a valid date format. |
| *isTimeFormat()* | Checks if string is a valid time format. |
| *validateForm()* | Checks if all the elements of a form have valid values. |
| *passFilter()* | Acording to the value of 'contains', checks if all the characters of 'cad' are contained in 'filter' or if neither of the characters are contained. |

***Parsing functions***

| | |
|---|---|
| *parseNumber()* | Parses a string and returns an number. |
| *parseInteger()* | Parses a string and returns a integer. |
| *parseIPv4()* | Parses a string that represents an IPv4 address and returns a array with four numbers. |
| *parseIPv6()* | Parses a string that represents an IPv6 address and returns a array with eight hexadecimal numbers. |
| *parseDate()* | Receives a string representing a date and transforms it according to the format passed as parameter. |
| *parseTime()* | Receives a string representing an instant of time and transforms it according to the format passed as parameter. |
| *filterString()* | Returns all the characters of 'cad' that are contained, or that aren't contained, in 'filter'. |

***Others***

| | |
|---|---|
| *compareDate()* | Compare two strings that represents dates. |
| *compareTime()* | Compare two strings that represents times. |
| *serialize()* | Encode the elements of a form as a string. |

### 7.1.2 - About Seahorse

*Introduction*

Seahorse is an open source JavaScript library created for simplify the use of forms, particularly the form validation.

It provides functions to validate, parse and serialize data and to add real-time validation to inputs.

Seahorse can be used alone or it can be used with jQuery, using the plugin designed for that purpose.

*License*

This library is licensed under the LGPL Version 3 license.

That means you can use Seahorse to develop commercial or open source projects, but, in both cases, you must publish, under a licence compatible with LGPL v3, any changes you make to the library.

### 7.1.3 - Validation behaviors

One of the main features of this library are the functions to assign validation behaviors to the inputs.

An input with a validation behavior can avoid the input of invalid characters or perform diferent actions (like add a CSS class to a element or invoke a JavaScript function) when a user inserts valid or invalid data.

The validation behaviors functions receives as parameter the id of the element to be validated, a JSON element with the validation options and a JSON element with the response options.

*Validation options*

The validation options are parameters that restrict the values, or the format, that the data of an input can have to be considered valid.

The variables that can be defined in the JSON element of validation options are:

| | |
|---|---|
| *notEmpty* | A boolean indicating if the field can be left in blank. |
| *minLength* | The minimum length of a text input (by default, 0). |
| *maxLength* | The maximun length of a text input (by default, infinity). |
| *minValue* | The minimum value of a input (by default, minus infinity for numbers and null for dates and times). |
| *maxValue* | The maximun value of a input (by default, infinity for numbers and null for dates and times). |
| *format* | The format of a input (by default 'yyyy-mm-dd' for dates and 'hh:mm:ss' for times). |
| *autofill* | A boolean indicating if the incomplete fields of dates or times must be completed with the actual date or time. |
| *version* | The version of an IP address input (by default, 4). |
| *requiredCharacters* | A string or a array of unicode values representing the group of characters that a text input must have to be considered valid. |
| *forbiddenCharacters* | A string or a array of unicode values representing the group of characters that a text input must not have to be considered valid. |
| *allowedCharacters* | A string or a array of unicode values representing the group of characters that a text input can have and be considered valid, no matter the restrictions of his type. |
| *asciiCharacters* | A boolean indicating if a text input must have only ASCII characters (by default 'false'). |
| *decimalCharacter* | The character used as decimal character in a numeric input (by default '.'). |
| *groupingCharacter* | The character used as grouping character in a numeric input (by default ','). |

| | |
|---|---|
| *aditionalValidation* | A user's defined function that test if a field has a valid value. This function is called only if the field has been validated by the standard Seahorse's validation function. |

This variables are optionals and they are not used by all the functions, each function uses only a few variables, for example, for restrict the lenght, text() uses minLength and maxLength while number() uses minValue and maxValue.

*Response options*

The response options are parameters that determines the courses of actions when an user enter a valid or invalid input.

The variables that can be defined in the JSON element of response options are:

| | |
|---|---|
| *errorClass* | The class added to the input if the data entered is invalid. |
| *okClass* | The class added to the input if the data entered is valid. |
| *targetErrorClass* | The class added to a given element if the data entered is invalid. |
| *targetOkClass* | The class added to a given element if the data entered is valid. |
| *targetId* | The id of the element to which add the classes 'targetErrorClass' or 'targetOkClass'. |
| *hiddenElementId* | The id of the element to hide or show, depending if the data entered is valid or invalid. |
| *callbackFunction* | A function to invoke after that the input was validated. The function must receive two parameters: 'element' (the object that represents the input) and 'valid' (a boolean indicating if the data entered is valid or not). |
| *forbidEntrance* | A boolean indicating if the entrance of invalid characters must be forbidden (by default, 'true'). |
| *autoparse* | A boolean indicating if the data in the fields must be parsed in order to eliminate unnecessary characters. |
| *errorMessage* | A message explaining why the value of the input is invalid. |

This variables are optionals and can be specified for all the validation functions.

### 7.1.4 - Functions

*Sumary*

***Validation behaviors***

| | |
|---|---|
| text() | Gives to an element a text input behavior. |
| alphabetical() | Gives to an element an alphabetical input behavior. |
| alphanumeric() | Gives to an element an alphanumeric input behavior. |
| numeric() | Gives to an element a numeric input behavior. |
| number() | Gives to an element a float input behavior. |
| integer() | Gives to an element an integer input behavior. |
| date() | Gives to an element a date input behavior. |
| time() | Gives to an element a time input behavior. |
| ipAddress() | Gives to an element an IP address input behavior. |
| email() | Gives to an element an e-mail address input behavior. |
| http() | Gives to an element a HTTP address input behavior. |
| ftp() | Gives to an element a FTP address input behavior. |
| form() | Gives to an element a form behavior. |
| removeBehavior() | Removes, from a element, any behavior asigned by a function of the Seahorse's library. |

***Validation functions***

| | |
|---|---|
| isNumber() | Checks if a string represents a number. |
| isInteger() | Checks if a string represents an integer. |
| isNumeric() | Checks if a string contains only numbers. |
| isAlphabetical() | Checks if a string contains only alphabetical characters (including accents and another not Ascii characters). |
| isAlphanumeric() | Checks if a string contains only alphanumeric characters. |
| isAlphabeticalAscii() | Checks if a string contains only alphabetical ASCII characters. |
| isAlphanumericAscii() | Checks if a string contains only alphanumeric ASCII characters. |
| isAsciiText() | Checks if a string contains only ASCII characters. |
| isIPv4() | Checks if a string represents an IP version 4 address. |
| isIPv6() | Checks if a string represents an IP version 6 address. |
| isEmail() | Checks if a string represents an e-mail address. |
| isHttp() | Checks if a string represents a HTTP address. |

| | |
|---|---|
| isFtp() | Checks if a string represents a FTP address. |
| isMonth() | Checks if a string represents a month. |
| isDay() | Checks if a string represents a day of a particular month. |
| isDate() | Checks if string represents a date. |
| isTime() | Checks if string represents an instant of time. |
| isDateFormat() | Checks if string is a valid date format. |
| isTimeFormat() | Checks if string is a valid time format. |
| validateForm() | Checks if all the elements of a form have valid values. |
| passFilter() | Acording to the value of 'contains', checks if all the characters of 'cad' are contained in 'filter' or if neither of the characters are contained. |

***Parsing functions***

| | |
|---|---|
| parseNumber() | Parses a string and returns an number. |
| parseInteger() | Parses a string and returns a integer. |
| parseIPv4() | Parses a string that represents an IPv4 address and returns a array with four numbers. |
| parseIPv6() | Parses a string that represents an IPv6 address and returns a array with eight hexadecimal numbers. |
| parseDate() | Receives a string representing a date and transforms it according to the format passed as parameter. |
| parseTime() | Receives a string representing an instant of time and transforms it according to the format passed as parameter. |
| filterString() | Returns all the characters of 'cad' that are contained, or that aren't contained, in 'filter'. |

***Others***

| | |
|---|---|
| compareDate() | Compare two strings that represents dates. |
| compareTime() | Compare two strings that represents times. |
| serialize() | Encode the elements of a form as a string. |

*Validation behaviors*

## text()

```
text : function(  element,
                  validationOptions,
                  responseOptions    )
```

Gives to an element a text input behavior.

*Parameters*

| | |
|---|---|
| element | The element id or the element object. |
| validationOptions | A JSON element with the validation options (notEmpty, aditionalValidation, minLength, maxLength, asciiCharacters, allowedCharacters, requiredCharacters and forbiddenCharacters). |
| responseOptions | A JSON element with the response options. |

## alphabetical()

```
alphabetical : function(  element,
                          validationOptions,
                          responseOptions    )
```

Gives to an element an alphabetical input behavior.

*Parameters*

| | |
|---|---|
| element | The element id or the element object. |
| validationOptions | A JSON element with the validation options (notEmpty, aditionalValidation, minLength, maxLength, asciiCharacters, allowedCharacters, requiredCharacters and forbiddenCharacters). |
| responseOptions | A JSON element with the response options. |

### alphanumeric()

```
alphanumeric : function(  element,
                          validationOptions,
                          responseOptions    )
```

Gives to an element an alphanumeric input behavior.

*Parameters*

| | |
|---|---|
| element | The element id or the element object. |
| validationOptions | A JSON element with the validation options (notEmpty, aditionalValidation, minLength, maxLength, asciiCharacters, allowedCharacters, requiredCharacters and forbiddenCharacters). |
| responseOptions | A JSON element with the response options. |

### numeric()

```
numeric : function(  element,
                     validationOptions,
                     responseOptions    )
```

Gives to an element a numeric input behavior.

*Parameters*

| | |
|---|---|
| element | The element id or the element object. |
| validationOptions | A JSON element with the validation options (notEmpty, aditionalValidation, minLength, maxLength, asciiCharacters, allowedCharacters, requiredCharacters and forbiddenCharacters). |
| responseOptions | A JSON element with the response options. |

### number()

```
number : function(  element,
                    validationOptions,
                    responseOptions    )
```

Gives to an element a float input behavior.

*Parameters*

| | |
|---|---|
| element | The element id or the element object. |
| validationOptions | A JSON element with the validation options (notEmpty, aditionalValidation, minValue, maxValue, groupingCharacter and decimalCharacter). |
| responseOptions | A JSON element with the response options. |

### integer()

```
integer : function(  element,
                     validationOptions,
                     responseOptions    )
```

Gives to an element an integer input behavior.

*Parameters*

| | |
|---|---|
| element | The element id or the element object. |
| validationOptions | A JSON element with the validation options (notEmpty, aditionalValidation, minValue, maxValue and groupingCharacter). |

| | |
|---|---|
| responseOptions | A JSON element with the response options. |

### date()

```
date : function(  element,
                  validationOptions,
                  responseOptions     )
```

Gives to an element a date input behavior.

*Parameters*

| | |
|---|---|
| element | The element id or the element object. |
| validationOptions | A JSON element with the validation options (notEmpty, aditionalValidation, minValue, maxValue, autofill and format). |
| responseOptions | A JSON element with the response options. |

### time()

```
time : function(  element,
                  validationOptions,
                  responseOptions     )
```

Gives to an element a time input behavior.

*Parameters*

| | |
|---|---|
| element | The element id or the element object. |
| validationOptions | A JSON element with the validation options (notEmpty, aditionalValidation, minValue, maxValue, autofill and format). |
| responseOptions | A JSON element with the response options. |

### ipAddress()

```
ipAddress : function(  element,
                       validationOptions,
                       responseOptions     )
```

Gives to an element an IP address input behavior.

*Parameters*

| | |
|---|---|
| element | The element id or the element object. |
| validationOptions | A JSON element with the validation options (notEmpty, aditionalValidation and version). |
| responseOptions | A JSON element with the response options. |

### email()

```
email : function(  element,
                   validationOptions,
                   responseOptions     )
```

Gives to an element an e-mail address input behavior.

*Parameters*

| | |
|---|---|
| element | The element id or the element object. |
| validationOptions | A JSON element with the validation options (notEmpty and aditionalValidation). |
| responseOptions | A JSON element with the response options. |

### http()

```
http : function(  element,
                  validationOptions,
                  responseOptions    )
```

Gives to an element a HTTP address input behavior.

*Parameters*

| | |
|---|---|
| element | The element id or the element object. |
| validationOptions | A JSON element with the validation options (notEmpty and aditionalValidation). |
| responseOptions | A JSON element with the response options. |

### ftp()

```
ftp : function(  element,
                 validationOptions,
                 responseOptions    )
```

Gives to an element a FTP address input behavior.

*Parameters*

| | |
|---|---|
| element | The element id or the element object. |
| validationOptions | A JSON element with the validation options (notEmpty and aditionalValidation). |
| responseOptions | A JSON element with the response options. |

### form()

```
form : function(  form,
                  responseFunction,
                  errorMessage       )
```

Gives to an element a form behavior.
That means that it modifies the submit method of the form in order to validate all the inputs before submit. The 'responseFunction' parameter is the function called when the form is submited but one or more inputs, of the form, has invalid values. This function must receive two parameters: a string with the error messages of the form and the inputs and an array with all the input elements that have invalid values.

*Parameters*

| | |
|---|---|
| form | The form id or the form object. |
| responseFunction | The function called if one of more inputs of the form are invalid. |
| errorMessage | The message to be displayed if one of more inputs of the form are invalid. |

### removeBehavior()

```
removeBehavior : function(    element )
```

Removes, from a element, any behavior asigned by a function of the Seahorse's library.

*Parameters*

element The element id or the element object.

### isNumber()

```
isNumber : function(  cad,
                      ds,
                      gs   )
```

Checks if a string represents a number.

*Parameters*
cad A string.
ds The character used as digital separator.
gs The character used as grouping separator.

*Returns*
`true` if the string represents a number and `false` in the opposite case.

### isInteger()

```
isInteger : function(  cad,
                       gs   )
```

Checks if a string represents an integer.

*Parameters*
cad A string.
gs The character used as grouping separator.

*Returns*
`true` if the string represents an integer and `false` in the opposite case.

### isNumeric()

```
isNumeric : function(  cad )
```

Checks if a string contains only numbers.

*Parameters*
cad A string.

*Returns*
`true` if the string contains only numbers and `false` in the opposite case.

### isAlphabetical()

```
isAlphabetical : function(  cad )
```

Checks if a string contains only alphabetical characters (including accents and another not Ascii characters).

*Parameters*
cad A string.

*Returns*
`true` if the string contains only alphabetical characters and `false` in the opposite case.

### isAlphanumeric()

isAlphanumeric : function( cad )

Checks if a string contains only alphanumeric characters.

*Parameters*
cad A string.

*Returns*
`true` if the string contains only alphanumeric characters and `false` in the opposite case.

### isAlphabeticalAscii()

isAlphabeticalAscii : function( cad )

Checks if a string contains only alphabetical ASCII characters.

*Parameters*
cad A string.

*Returns*
`true` if the string contains only alphabetical ASCII characters and `false` in the opposite case.

### isAlphanumericAscii()

isAlphanumericAscii : function( cad )

Checks if a string contains only alphanumeric ASCII characters.

*Parameters*
cad A string.

*Returns*
`true` if the string contains only alphanumeric ASCII characters and `false` in the opposite case.

### isAsciiText()

isAsciiText : function( cad )

Checks if a string contains only ASCII characters.

*Parameters*
cad A string.

*Returns*
`true` if the string contains only ASCII characters and `false` in the opposite case.

### isIPv4()

isIPv4 : function( ip )

Checks if a string represents an IP version 4 address.

*Parameters*
ip A string.

*Returns*
`true` if the string represents an IP version 4 address and `false` in the opposite case.

### isIPv6()
isIPv6 : function( ip )

Checks if a string represents an IP version 6 address.

*Parameters*
ip A string.

*Returns*
`true` if the string represents an IP version 6 address and `false` in the opposite case.

### isEmail()
isEmail : function( mail )

Checks if a string represents an e-mail address.

*Parameters*
mail A string.

*Returns*
`true` if the string represents an e-mail address and `false` in the opposite case.

### isHttp()
isHttp : function( http )

Checks if a string represents a HTTP address.

*Parameters*
http A string.

*Returns*
`true` if the string represents a HTTP address and `false` in the opposite case.

### isFtp()
isFtp : function( ftp )

Checks if a string represents a FTP address.

*Parameters*
ftp A string.

*Returns*
`true` if the string represents a FTP address and `false` in the opposite case.

### isMonth()
isMonth : function( mes )

Checks if a string represents a month. This function considers that a month is a number that must be between 1 and 12 (while the class Date considers a month like a number between 0 and 11).

*Parameters*
mes A string.

*Returns*
`true` if the string represents a month and `false` in the opposite case.

### isDay()

```
isDay : function(  dia,
                   mes,
                   anio )
```

Checks if a string represents a day of a particular month. This function considers that a month is a number that must be between 1 and 12 (while the class Date considers a month like a number between 0 and 11).

*Parameters*
dia A string that represents a day.
mes A string that represents a month.
anio A string that represents a year.

*Returns*
`true` if 'dia' represents a day of the month 'mes' and `false` in the opposite case or if 'mes' it's not a month.

### isDate()

```
isDate : function(  cad,
                    format,
                    fill       )
```

Checks if string represents a date.

*Parameters*
cad A string that represents a date.
format A string that represents a date format.
fill A boolean that indicates if the empty fields have to be completed with the actual date.

*Returns*
`true` if the string represents a date and `false` in the opposite case.

### isTime()

```
isTime : function(  cad,
                    format,
                    fill       )
```

Checks if string represents an instant of time.

*Parameters*
cad A string that represents an instant of time.
format A string that represents a time format.
fill A boolean that indicates if the empty fields have to be completed with the actual time.

*Returns*

`true` if the string represents an instant of time and `false` in the opposite case.

### isDateFormat()

isDateFormat : function( format )

Checks if string is a valid date format.

*Parameters*
cad A string that represents a date format.

*Returns*
`true` if the string is a valid date format and `false` in the opposite case.

### isTimeFormat()

isTimeFormat : function( format )

Checks if string is a valid time format.

*Parameters*
cad A string that represents a time format.

*Returns*
`true` if the string is a valid time format and `false` in the opposite case.

### validateForm()

validateForm : function( idForm )

Checks if all the elements of a form have valid values.

*Parameters*
idForm The id of the form to be validated.

*Returns*
`true` if all the elements of the form have valid values. `false` if one or more elements of the form have invalid values.

### passFilter()

passFilter : function( cad,
filter,
contains )

Acording to the value of 'contains', checks if all the characters of 'cad' are contained in 'filter' or if neither of the characters are contained.

*Parameters*
cad A string to filter.
filter A string used as filter.
contains A boolean indicating the mode of operation.

*Returns*
`true` if all the characters of 'cad' are contained in 'filter' and `false` in the oposite case. `true` if neither of the characters of 'cad' are contained in 'filter' and `false` in the opposite case.

*Parsing functions*

### parseNumber()

```
parseNumber : function(  cad,
                         ds,
                         gs   )
```

Parses a string and returns an number.

*Parameters*
cad A string.
ds The character used as digital separator.
gs The character used as grouping separator.

Returns
A number if the string is a number or `NaN` in the opposite case.

### parseInteger()

```
parseInteger : function(  cad,
                          gs   )
```

Parses a string and returns a integer.

*Parameters*
cad A string.
gs The character used as grouping separator.

*Returns*
An integer if the string is a number or `NaN` in the opposite case.

### parseIPv4()

```
parseIPv4 : function(  ip )
```

Parses a string that represents an IPv4 address and returns a array with four numbers.

*Parameters*
*ip* A string.

*Returns*
An array of four numbers if the string represents an IPv4 address or `null` in the opposite case.

### parseIPv6()

```
parseIPv6 : function(  ip )
```

Parses a string that represents an IPv6 address and returns a array with eight hexadecimal numbers.

*Parameters*
*ip* A string.

*Returns*
An array of eight hexadecimal numbers if the string represents an IPv6 address or `null` in the opposite case.

### parseDate()

```
parseDate : function(  cad,
                       format,
                       fill       )
```

Receives a string representing a date and transforms it according to the format passed as parameter.

*Parameters*
cad A string that represents a date.
format A string that represents a valid date format.
fill A boolean that indicates if the empty fields have to be completed with the actual date.

*Returns*
A formated string that represents a date if 'cad' represents a date and 'format' is a valid date format or `null` in the opposite case.

### parseTime()

```
parseTime : function(  cad,
                       format,
                       fill       )
```

Receives a string representing an instant of time and transforms it according to the format passed as parameter.

*Parameters*
cad A string that represents an instant of time.
format A string that represents a valid time format.
fill A boolean that indicates if the empty fields have to be completed with the actual time.

*Returns*
A formated string that represents an instant of time if 'cad' represents an instant of time and 'format' is a valid time format or `null` in the opposite case.

### filterString()

```
filterString : function(  cad,
                          filter,
                          contains )
```

Returns all the characters of 'cad' that are contained, or that aren't contained, in 'filter'.

*Parameters*
cad A string to filter.
filter A string used as filter.
contains A boolean indicating the mode of operation.

*Returns*
All the characters of 'cad' that are contained in 'filter', if 'contains' is equal to `true`. All the characters of 'cad' that aren't contained in 'filter', if 'contains' is equal to `false`.

*Others*

### compareDate()

```
compareDate : function(  date1,
```

```
                                date2,
                                dateFormat )
```

Compare two strings that represents dates.

*Parameters*

| | |
|---|---|
| date1 | The first date. |
| date2 | The second date. |
| dateFormat | The date format of the two dates. |

*Returns*

A negative number if date1 < date2, a positive number if date1 > date2, zero if date1 = date2 or null in case of error.

### compareTime()

```
compareTime : function(   time1,
                          time2,
                          timeFormat  )
```

Compare two strings that represents times.

*Parameters*

| | |
|---|---|
| time1 | The first time. |
| time2 | The second time. |
| timeFormat | The date format of the two times. |

*Returns*

A negative number if time1 < time2, a positive number if time1 > time2, zero if time1 = time2 or null in case of error.

### serialize()

```
serialize : function(   form,
                        notation )
```

Encode the elements of a form as a string.

*Parameters*

| | |
|---|---|
| form | The form id or the form object. |
| codification | The notation to be used (JSON or URL) |

*Returns*

The form serialized.

## 7.2 - Examples

### 7.2.1 - Validation

```
<html>

<head>
<title>Seahorse's examples - Validation</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<script type="text/javascript" src="js/seahorse-1.2.js"></script>

</head>
<body>

   <table>
      <tr>
         <td>Seahorse.isNumber("1.000.000,00", ',', '.')</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.isNumber("1.000.000,00", ',', '.'));
            </script>
         </td>
      </tr>
      <tr>
         <td>Seahorse.isInteger("1,000,000", ',')</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="=" /></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.isInteger("1,000,000", ','));
            </script>
         </td>
      </tr>
      <tr>
         <td>Seahorse.isNumeric("1234")</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.isNumeric("1234"));
            </script>
         </td>
      </tr>
      <tr>
         <td>Seahorse.isAlphabetical("abcd&ntilde;")</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.isAlphabetical("abcd\u00c9"));
            </script>
         </td>
      </tr>
      <tr>
         <td>Seahorse.isAlphanumeric("abcd&ntilde;123")</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.isAlphanumeric("abcd\u00c9123"));
            </script>
         </td>
      </tr>
      <tr>
         <td>Seahorse.isAlphabeticalAscii("abcd")</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.isAlphabeticalAscii("abcd"));
            </script>
         </td>
```

```
</tr>
<tr>
   <td>Seahorse.isAlphanumericAscii("abc123")</td>
   <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
   <td>
      <script type="text/javascript">
         document.write(Seahorse.isAlphanumericAscii("abcd"));
      </script>
   </td>
</tr>
<tr>
   <td>Seahorse.isAsciiText("abcd (123) @#")</td>
   <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
   <td>
      <script type="text/javascript">
         document.write(Seahorse.isAsciiText("abcd"));
      </script>
   </td>
</tr>
<tr>
   <td>Seahorse.isIPv4("192.168.0.1")</td>
   <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
   <td>
      <script type="text/javascript">
         document.write(Seahorse.isIPv4("192.168.0.1"));
      </script>
   </td>
</tr>
<tr>
   <td>Seahorse.isIPv6("11:22:33:44:55:66:77:88")</td>
   <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
   <td>
      <script type="text/javascript">
         document.write(Seahorse.isIPv6("11:22:33:44:55:66:77:88"));
      </script>
   </td>
</tr>
<tr>
   <td>Seahorse.isEmail("test@test.com")</td>
   <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
   <td>
      <script type="text/javascript">
         document.write(Seahorse.isEmail("test@test.com"));
      </script>
   </td>
</tr>
<tr>
   <td>Seahorse.isHttp("http://www.test.com")</td>
   <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
   <td>
      <script type="text/javascript">
         document.write(Seahorse.isHttp("http://www.test.com"));
      </script>
   </td>
</tr>
<tr>
   <td>Seahorse.isFtp("ftp://ftp.test.com")</td>
   <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
   <td>
      <script type="text/javascript">
         document.write(Seahorse.isFtp("ftp://ftp.test.com"));
      </script>
   </td>
</tr>
<tr>
   <td>Seahorse.isDate("31/07/2010","dd/mm/yyyy")</td>
   <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
   <td>
```

```
            <script type="text/javascript">
               document.write(Seahorse.isDate("31/07/2010","dd/mm/yyyy"));
            </script>
         </td>
      </tr>
      <tr>
         <td>Seahorse.isTime("06:20:22","hh:mm:ss")</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.isTime("06:20:22","hh:mm:ss"));
            </script>
         </td>
      </tr>
   </table>

</body>
</html>
```

### 7.2.2 - Parsing

```
<html>

<head>
<title>Seahorse's examples - Parsing</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<script type="text/javascript" src="js/seahorse-1.2.js"></script>

</head>
<body>

   <table>
      <tr>
         <td>Seahorse.parseNumber("1,234.5", '.', ',')</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.parseNumber("1,234.5", '.', ','));
            </script>
         </td>
      </tr>
      <tr>
         <td>Seahorse.parseInteger("1,234", ',')</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.parseInteger("1,234", ','));
            </script>
         </td>
      </tr>
      <tr>
         <td>Seahorse.parseIPv4("192.168.0.1")</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.parseIPv4("192.168.0.1"));
            </script>
         </td>
      </tr>
      <tr>
         <td>Seahorse.parseIPv6("11:22:33:44:55:66:77:88")</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.parseIPv6("11:22:33:44:55:66:77:88"));
            </script>
         </td>
      </tr>
```

```
      <tr>
         <td>Seahorse.parseDate("31/07/2010","dd/mm/yyyy")</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.parseDate("31/07/2010","dd/mm/yyyy"));
            </script>
         </td>
      </tr>
      <tr>
         <td>Seahorse.parseTime("02:05:33","hh:mm:ss")</td>
         <td><img src="img/icon-arrow-right.png" title="=" alt="="/></td>
         <td>
            <script type="text/javascript">
               document.write(Seahorse.parseTime("02:05:33","hh:mm:ss"));
            </script>
         </td>
      </tr>
   </table>

</body>
</html>
```

### 7.2.3 - Behaviors

```
<html>

<head>
<title>Seahorse's examples - Parsing</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      Seahorse.number( "number1", {"decimalCharacter": '.', "groupingCharacter" : ','},
{ "okClass": "ok", "errorClass": "error" } );
      document.getElementById("number1").seahorse.verify();

      Seahorse.integer( "integer1", {"groupingCharacter" : ','}, { "okClass": "ok",
"errorClass": "error" } );
      document.getElementById("integer1").seahorse.verify();

      Seahorse.date( "date1", {"format" : 'dd/mm/yyyy', 'autofill': true}, { "okClass":
"ok", "errorClass": "error" } );
      document.getElementById("date1").seahorse.verify();

      Seahorse.time( "time1", {"format" : 'hh:mm:ss', 'autofill': true}, { "okClass":
"ok", "errorClass": "error" } );
      document.getElementById("time1").seahorse.verify();

      Seahorse.ipAddress( "ipv41", {"version" : 4 }, { "okClass": "ok", "errorClass":
"error" } );
      document.getElementById("ipv41").seahorse.verify();

      Seahorse.email( "email1", {}, { "okClass": "ok", "errorClass": "error" } );
      document.getElementById("email1").seahorse.verify();

      Seahorse.http( "http1", {}, { "okClass": "ok", "errorClass": "error" } );
      document.getElementById("http1").seahorse.verify();

      Seahorse.ftp( "ftp1", {}, { "okClass": "ok", "errorClass": "error" } );
      document.getElementById("ftp1").seahorse.verify();
```

```
     Seahorse.text( "text1", {}, { "okClass": "ok", "errorClass": "error" } );
     document.getElementById("text1").seahorse.verify();

     Seahorse.alphabetical( "alphabetical1", {}, { "okClass": "ok", "errorClass": "error"
} );
     document.getElementById("alphabetical1").seahorse.verify();

     Seahorse.alphanumeric( "alphanumeric1", {}, { "okClass": "ok", "errorClass": "error"
} );
     document.getElementById("alphanumeric1").seahorse.verify();

     Seahorse.numeric( "numeric1", {}, { "okClass": "ok", "errorClass": "error" } );
     document.getElementById("numeric1").seahorse.verify();
     }
</script>

</head>
<body onload="javascript:init();">

   <table>
     <tr>
        <td>number()</td>
        <td><input type="text" id="number1" value="1,000.00"/></td>
     </tr>
     <tr>
        <td>integer()</td>
        <td><input type="text" id="integer1" value="1,000"/></td>
     </tr>
     <tr>
        <td>date()</td>
        <td><input type="text" id="date1" value="31/12/2000"/></td>
     </tr>
     <tr>
        <td>time()</td>
        <td><input type="text" id="time1" value="23:59:59"/></td>
     </tr>
     <tr>
        <td>ipAddress()</td>
        <td><input type="text" id="ipv41" value="192.168.0.1"/></td>
     </tr>
     <tr>
        <td>email()</td>
        <td><input type="text" id="email1" value="test@server.com"/></td>
     </tr>
     <tr>
        <td>http()</td>
        <td><input type="text" id="http1" value="http://www.test.com"/></td>
     </tr>
     <tr>
        <td>ftp()</td>
        <td><input type="text" id="ftp1" value="ftp://ftp.test.com"/></td>
     </tr>
     <tr>
        <td>text()</td>
        <td><input type="text" id="text1" value="Text"/></td>
     </tr>
     <tr>
        <td>alphanumeric()</td>
        <td><input type="text" id="alphanumeric1" value="Alphanumeric123"/></td>
     </tr>
     <tr>
        <td>alphabetical()</td>
        <td><input type="text" id="alphabetical1" value="Alphabetical"/></td>
     </tr>
     <tr>
        <td>numeric()</td>
        <td><input type="text" id="numeric1" value="1234"/></td>
```

```
      </tr>
   </table>

</body>
</html>
```

### 7.2.4 - jQuery

```
<html>

<head>
<title>Seahorse's examples - Parsing</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<style type="text/css">
   .ok { background-color:#f0fff0; }
   .error { background-color:#fff0f0; }
</style>

<script type="text/javascript" src="js/jquery-1.4.2.js"></script>
<script type="text/javascript" src="js/seahorse-1.2.js"></script>
<script type="text/javascript" src="js/seahorse.jquery-1.2.js"></script>
<script type="text/javascript">
   function init()
      {
      jQuery(".number").seaBehavior( "number", {"decimalCharacter": '.',
"groupingCharacter" : ','}, { "okClass": "ok", "errorClass": "error" } );
      jQuery(".number").seaVerify();

      jQuery(".integer").seaBehavior("integer", {"groupingCharacter" : ','}, { "okClass":
"ok", "errorClass": "error" } );
      jQuery(".integer").seaVerify();

      jQuery(".date").seaBehavior( "date", {"format" : 'dd/mm/yyyy', 'autofill': true},
{ "okClass": "ok", "errorClass": "error" } );
      jQuery(".date").seaVerify();

      jQuery(".time").seaBehavior( "time", {"format" : 'hh:mm:ss', 'autofill': true},
{ "okClass": "ok", "errorClass": "error" } );
      jQuery(".time").seaVerify();

      jQuery(".ipAddress")
         .seaBehavior( "ipAddress", {"version" : 4 }, { "okClass": "ok", "errorClass":
"error" } )
         .seaVerify();

      jQuery(".email")
         .seaBehavior( "email", {}, { "okClass": "ok", "errorClass": "error" } )
         .seaVerify();

      jQuery(".http")
         .seaBehavior( "http", {}, { "okClass": "ok", "errorClass": "error" } )
         .seaVerify();

      jQuery(".ftp")
         .seaBehavior( "ftp", {}, { "okClass": "ok", "errorClass": "error" } )
         .seaVerify();

      jQuery(".text")
         .seaBehavior( "text", {}, { "okClass": "ok", "errorClass": "error" } )
         .seaVerify();

      jQuery(".alphabetical")
         .seaBehavior( "alphabetical", {}, { "okClass": "ok", "errorClass": "error" } )
         .seaVerify();

      jQuery(".alphanumeric")
         .seaBehavior( "alphanumeric", {}, { "okClass": "ok", "errorClass": "error" } )
```

```
        .seaVerify();

     jQuery(".numeric")
        .seaBehavior( "numeric", {}, { "okClass": "ok", "errorClass": "error" } )
        .seaVerify();
     }
</script>

</head>
<body onload="javascript:init();">

   <table>
     <tr>
        <td>number()</td>
        <td><input type="text" class="number" value="1,000.00"/></td>
     </tr>
     <tr>
        <td>integer()</td>
        <td><input type="text" class="integer" value="1,000"/></td>
     </tr>
     <tr>
        <td>date()</td>
        <td><input type="text" class="date" value="31/12/2000"/></td>
     </tr>
     <tr>
        <td>time()</td>
        <td><input type="text" class="time" value="23:59:59"/></td>
     </tr>
     <tr>
        <td>ipAddress()</td>
        <td><input type="text" class="ipAddress" value="192.168.0.1"/></td>
     </tr>
     <tr>
        <td>email()</td>
        <td><input type="text" class="email" value="test@server.com"/></td>
     </tr>
     <tr>
        <td>http()</td>
        <td><input type="text" class="http" value="http://www.test.com"/></td>
     </tr>
     <tr>
        <td>ftp()</td>
        <td><input type="text" class="ftp" value="ftp://ftp.test.com"/></td>
     </tr>
     <tr>
        <td>text()</td>
        <td><input type="text" class="text" value="Text"/></td>
     </tr>
     <tr>
        <td>alphanumeric()</td>
        <td><input type="text" class="alphanumeric" value="Alphanumeric123"/></td>
     </tr>
     <tr>
        <td>alphabetical()</td>
        <td><input type="text" class="alphabetical" value="Alphabetical"/></td>
     </tr>
     <tr>
        <td>numeric()</td>
        <td><input type="text" class="numeric" value="1234"/></td>
     </tr>
   </table>

</body>
</html>
```

## 7.3 - License legal code


**Attribution-NonCommercial-NoDerivs 3.0 Unported**

CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE LEGAL SERVICES. DISTRIBUTION OF THIS LICENSE DOES NOT CREATE AN ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO WARRANTIES REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS LIABILITY FOR DAMAGES RESULTING FROM ITS USE.

*License*

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

**1. Definition**

a. **"Adaptation"** means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.

b. **"Collection"** means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined above) for the purposes of this License.

c. **"Distribute"** means to make available to the public the original and copies of the Work through sale or other transfer of ownership.

d. **"Licensor"** means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.

e. **"Original Author"** means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.

f. **"Work"** means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.

g. **"You"** means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

h. **"Publicly Perform"** means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.

i. **"Reproduce"** means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

**2. Fair Dealing Rights.** Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

**3. License Grant.** Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

a. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections; and,

b. to Distribute and Publicly Perform the Work including as incorporated in Collections.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats, but otherwise you have no rights to make Adaptations. Subject to 8(f), all rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights set forth in Section 4(d).

**4. Restrictions.** The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section

4(c), as requested.

b. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.

c. If You Distribute, or Publicly Perform the Work or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work. The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Collection, at a minimum such credit will appear, if a credit for all contributing authors of Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.

d. For the avoidance of doubt:

    i. Non-waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;

    ii. Waivable Compulsory License Schemes. In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License if Your exercise of such rights is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(b) and otherwise waives the right to collect royalties through any statutory or compulsory licensing scheme; and,

    iii. Voluntary License Schemes. The Licensor reserves the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License that is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(b).

e. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation.

**5. Representations, Warranties and Disclaimer**

UNLESS OTHERWISE MUTUALLY AGREED BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

**6. Limitation on Liability.**

EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR

EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

**7. Termination**

a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

**8. Miscellaneous**

a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

b. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

c. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

d. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

e. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, Creative Commons does not authorize the use by either party of the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time. For the avoidance of doubt, this trademark restriction does not form part of this License.

Creative Commons may be contacted at http://creativecommons.org/.