

Kreditkartennummern mit PHP validieren und dabei die erfassten Formulardaten erhalten

Dr. Volker Thormählen, 12. Juni 2018

Inhaltsverzeichnis

Abbildungsverzeichnis	II
Tabellenverzeichnis	II
Verzeichnis der Listings	II
1 Aufgabenstellung	1
2 Aussehen des Spendenformulars	1
3 Anweisungen im externen Style Sheet	4
4 Besonderheiten des Spendenformulars	6
4.1 Steuerelemente in der Feldgruppe Spende	6
4.2 Steuerelemente in der Feldgruppe Kreditkarte	7
4.2.1 Kartenmarke	8
4.2.2 Karteninhaber	8
4.2.3 Kartennummer	9
4.2.4 Ablaufdatum	9
4.2.5 Kartensicherheitscode	9
4.3 Erfasste Formulardaten nach der Validierung erneut anzeigen	10
5 Plausibilitätskontrolle des Spendenformulars	11
5.1 Plausibilität zwischen Land und Postleitzahl	11
5.2 Existenzkontrolle der Hausnummer im Textfeld 'strasse'	12
5.3 Gültigkeit der Eingaben in die Textfelder für Vor- und Nachname	13
5.4 Gültigkeitsprüfung einer Kreditkarte	13
5.5 Testdaten für die PHP-Funktion 'checkCreditCard'	18
5.6 Metadaten für die Gültigkeitsprüfung von Kreditkarten	20
6 Auswertung der Daten des Spendenformulars	20
7 Literatur- und Quellenverzeichnis	III

Abbildungsverzeichnis

Abb. 1: Aussehen des Spendenformulars mit Platzhaltern bzw. Standardwerten (Quelle: In Anlehnung an [1] und [2])	2
Abb. 2: Fehlermeldung (rot) bei mangelnder Plausibilität zwischen Kartenmarke und Kartensicherheitscode.....	10
Abb. 3: Fehlermeldungen (rot) bei mangelnder Plausibilität der Formular Daten	11
Abb. 4: PHP-Skript zur Validierung von Namen	13

Tabellenverzeichnis

Tab 1: Komponenten des Beispiels.....	1
Tab 2: Steuerelemente des Spendenformulars	3
Tab 3: Attribute der Steuerelemente des Spendenformulars	3
Tab 4: Kreditkartenmarken und Länge der Kartennummer	8

Verzeichnis der Listings

Listing 1: Style Sheet-Anweisungen für 'spendenFormular.php'	5
Listing 2: Steuerelement für Spendenhöhe in der Feldgruppe Spende mit PHP erzeugen	6
Listing 3: Steuerelement für Spendenrhythmus in der Feldgruppe Spenden mit PHP erzeugen.....	6
Listing 4: Steuerelemente in der Feldgruppe Kreditkarte.....	8
Listing 5 : Plausibilitätskontrolle zwischen Kartenmarke und Kartensicherheitscode (cvc)	10
Listing 6: Plausibilitätskontrolle zwischen Land und Länge der Postleitzahl (plz)	11
Listing 7: Hausnummer im Textfeld 'strasse' finden	12
Listing 8: PHP-Code zur Validierung einer Kreditkarte mittels Metadaten	17
Listing 9: Test-Kreditkartennummern bekannter Kreditkartenmarken (Quelle: [5], [6])	18
Listing 10: Überprüfte Kreditkartennummern von vier bekannten Kreditkartenmarken	19
Listing 11: Metadaten für die Gültigkeitsprüfung von Kreditkarten ('defCreditCards.php')	20

1 Aufgabenstellung

Am Beispiel eines Spendenformulars wird im Folgenden beschrieben, wie die Kreditkartennummern bekannter Kreditkartenmarken auf Richtigkeit überprüft und zugleich die eingegebenen Formulareingaben nach dem Absenden beibehalten werden können, um diese ggf. gezielt zu berichtigen.

Zur Anwendung kommen:

- Die Auszeichnungssprache HTML¹ zur inhaltlichen Strukturierung des Spendenformulars
- Die Formatierungssprache CSS² zur stilistischen Gestaltung des Spendenformulars
- Die Skriptsprache PHP zur Dynamisierung des Spendenformulars und zur Validierung der Eingaben eines Benutzers.

Vorausgesetzt werden Grundkenntnisse in HTML, CSS und PHP. Struktur und Gestaltung des Spendenformulars werden lediglich präsentiert. Ausführlich beschrieben und kommentiert werden allerdings viele der angewandten PHP-Skripte.

Für das Beispiel werden insgesamt 5 Skripte bzw. Dateien eingesetzt:

	Skript/Datei	... benötigt ³ ...	Kurzbeschreibung
1	spendenFormular.php	2, 4	Spendenformular strukturieren
2	spendenFormular.css		Spendenformular formatieren
3	checkCreditCard.php	4	Kreditkarte validieren
4	defineCreditCards.php		Metadaten zur Validierung bekannter Kreditkartenmarken
5	cc_checker.php.php	3	Kreditkartennummern validieren mit <i>checkCreditCard.php</i>

Tab 1: Komponenten des Beispiels

2 Aussehen des Spendenformulars

Aufbau und Gestaltung des Spendenformulars werden in Abb. 1 präsentiert. Es ist klar gegliedert in 5 Feldgruppen:

- *Hinweise*
- *Spender*
- *Spende*
- *Kreditkarte*
- *Zustimmung*

Die Feldgruppe „Kreditkarte“ steht im Mittelpunkt der weiteren Ausführungen. Vorher wird noch der Inhalt der zugehörigen CSS-Formatierungen präsentiert, siehe Listing 1.

Inhaltlich orientiert sich das HTML-Formular an zwei praktischen Beispielen:

- Formulare für Benutzereingaben, Abb. 4-2, in [1]
- Formulare schön gestalten, Abb. 16.1, in [2], S. 186 ff

¹ HTML steht für engl. *Hypertext Markup Language*. Es ist eine Auszeichnungssprache zur Strukturierung von Webseiten.

² CSS steht für engl. *Cascading Style Sheets*. Es ist eine Formatierungssprache zur Gestaltung des Inhalts einer HTML-Webseite.

³ Realisiert durch *include* Anweisung im jeweiligen PHP-Code.

Spendenformular

Hinweise

Wir freuen uns, dass Sie spenden möchten!
 Bitte geben Sie Ihren Namen, Ihre Adresse, die Spendenhöhe in € und Ihre Kreditkartennummer ein.
 Drücken Sie dann auf die Schaltfläche "Spende abschicken"
 Mit einem hochgestellten roten Stern (Asterisk) markierte Felder (sog. Pflichtfelder) müssen bedient werden.

Spender

Anrede * **Frau** **Herr**

Vorname *

Nachname *

Ihr Foto:

Str. mit HausNr. *

Wohnort *

Postleitzahl *

Land * ▾

Spende

Spendenhöhe * ▾

Spendenrhythmus * ▾

Kreditkarte

Kartenmarke * ▾

Karteninhaber *

Kartennummer *

Ablaufdatum * ▾ ▾

MM / JJ

Sicherheitscode *

Zustimmung

Dürfen wir Ihr Foto ggf. veröffentlichen?

Erlaubnis * **Ja** **Nein**

Abb. 1: Aussehen des Spendenformulars mit Platzhaltern bzw. Standardwerten (Quelle: In Anlehnung an [1] und [2])

Steuerelemente des Spendenformulars			
Feldgruppe	Feldtyp	Feldname	Bezeichnungsfeld
Hinweise	Listenfeld		Hinweise
Spender	Optionsgruppe (Pflichtfeld)	frau	Anrede
		herr	
	einzeilige Textfelder (Pflichtfelder)	vorname	Vorname
		nachname	Nachname
		strasse	Straße
		wohntort	Wohnort
		plz	Postleitzahl
Auswahlliste (Optionen)	land	Land	
Eingabefeld zum Hochladen eines Fotos	spenderFoto	Ihr Foto	
Spende	Auswahlliste (Optionen)	spendenBetrag	Spendenhöhe
		spendenRhythmus	Spendenrhythmus
Kreditkarte	Auswahlliste (Optionen)	kartenMarke	Kartenmarke
	einzeiliges Textfeld (Pflichtfeld)	kartenInhaber	Karteninhaber
		kartenNummer	Kartennummer
	Auswahlliste (Optionen)	ablaufMM	Ablaufdatum
		ablaufJJ	
einzeiliges Textfeld (Pflichtfeld)	cvc	Kartensicherheitscode	
Zustimmung	Optionsgruppe (Pflichtfeld)	erlaubnis	Bitte entscheiden, ...
*	Schaltfläche (Button)	abschicken	Spende abschicken

Tab 2: Steuerelemente des Spendenformulars

Steuerelemente des Spendenformulars	
Feldname	Attribute
anrede	class
frau	type="radio", id, name, checked, value="frau"
herr	type="radio", id, name, checked, value="herr"
vorname	type="text", name, size=30, required, placeholder, value
nachname	type="text", name, size=30, checked, placeholder, value
spenderFoto	type="file"
strasse	type="text", name, size=30, required, placeholder, value
wohntort	type="text", name, size=30, required, placeholder, value
plz	type="number", name, size=5, required, placeholder, value
land	Optionsliste, required, value
spendenBetrag	Optionsliste, name, required, option value
spendenRhythmus	Optionsliste, name, required, option value
kartenMarke	Optionliste, id, name, required, option value
kartenInhaber	type="text", style, name, size, required, placeholder, title, value
kartenNummer	type="text", name, size, minlength, maxlength, required, placeholder, title, value
ablaufMM	Optionsliste, id, name, required, option value
ablaufJJ	Optionsliste, id, name, required, option value
cvc	type="number", name, size=4, minlength, maxlength, required, pattern, placeholder, title, value
erlaubnis	type="radio", id, name, checked, required, value
abschicken	Schaltfläche, type="submit", name

Tab 3: Attribute der Steuerelemente des Spendenformulars

3 Anweisungen im externen Style Sheet

Zur Formatierung des Spendenformulars werden die in Listing 1 enthaltenen CSS-Anweisungen benutzt:

```
/* -----  
spendenformular.php mit CSS gestalten  
Datei: spendenformular.css  
Datum: 23. Mai 2018  
Autor: Dr. Tho.  
-----*/  
  
html, body {  
    padding:0;  
    border:0px none;  
}  
  
form {  
    width:660px;  
    margin:0 auto;  
}  
  
h1 {  
    background-color:#FC0;  
    font-size:14pt;  
    text-align:center;  
    margin:0px;  
    width:630px;  
}  
  
fieldset {  
    -moz-border-radius:7px;  
    border:1px solid black;  
    padding:10px;  
    width:610px;  
    margin-top:5px;  
    margin-bottom:5px;  
}  
  
fieldset ul {  
    list-style:none;  
    padding-left:0;  
}  
  
fieldset button {  
    background-color:yellow;  
    margin:auto;  
    display:block;  
}  
  
fieldset legend {  
    width:90px;  
    border:1px solid black;  
    color:black;  
    background-color:yellow;  
    font-family:Verdana;  
    font-weight:bold;  
    font-size:13px;  
    font-style:italic;  
    padding-right:5px;  
    padding-left:5px;  
    padding-top:2px;  
    padding-bottom:2px;  
    -moz-border-radius:3px;  
}  
}
```

```
fieldset label {
    float:left;
    width:155px;
    padding-left:20px;
    margin:5px;
    text-align:left;
    font-weight:bold;
}
fieldset input, fieldset select {
    margin:5px;
    padding:0px;
    float:left;
}
fieldset label sup {
    color:red;
    font-weight:bold;
}
fieldset span {
    font-size:80%;
    font-weight:bold;
}
fieldset br {
    clear:left;
}
fieldset p {
    font-weight:bold;
}
.error {
    color:red;
}
```

Listing 1: Style Sheet-Anweisungen für 'spendenFormular.php'

In [2] und ähnlichen Veröffentlichungen werden die in Listing 1 enthaltenen CSS-Anweisungen im Detail erläutert.

4 Besonderheiten des Spendenformulars

4.1 Steuerelemente in der Feldgruppe Spende

Die Feldgruppe *Spende* enthält zwei Steuerelemente (siehe Abb. 1)

1. *Spendenhöhe*
2. *Spendenrhythmus*

Die Auswahlliste für *Spendenhöhe* wird mit PHP in einer **for**-Schleife mit einer Schrittweite von 5 Euro erzeugt, siehe Listing 2.

```
<label for "spendenBetrag">Spendenhöhe<sup>*</sup></label>
<select name="spendenBetrag" size="1" required="required">
  <?php
    $obolus = "";
    for($i=5;$i<101;$i=$i+5){
      $obolus .= $i . ',';
    }
    $obolus = substr($obolus, 0, -1);
    $betrag = explode(",", $obolus);
    $spendenBetrag = "";
    if(isset($_POST['spendenBetrag'])){
      $spendenBetrag = $_POST['spendenBetrag'];
    }
    foreach($betrag as $option){
      echo '<option value="' . $option . '"';
      if ($option == $spendenBetrag){
        echo " selected";
      }
      echo ">" . $option . "€" . "</option>";
    }
  ?>
</select><br>
<span class="error"><?php echo $spendenBetragErr;?></span>
<br>
```

Listing 2: Steuerelement für Spendenhöhe in der Feldgruppe Spende mit PHP erzeugen

Die Auswahlliste für *Spendenrhythmus* wird ebenfalls mit PHP erstellt, siehe Listing 3.

```
<label for "spendenRhythmus">Spendenrhythmus<sup>*</sup>&nbsp;</label>
<select name="spendenRhythmus" required="required">
  <?php
    $rhythmus = array("einmalig", "wöchentlich", "monatlich", "jährlich");
    $spendenRhythmus = "";
    if(isset($_POST['spendenRhythmus'])){
      $spendenRhythmus = $_POST['spendenRhythmus'];
    }
    foreach($rhythmus as $option){
      echo '<option value="' . $option . '"';
      if ($option == $spendenRhythmus){
        echo " selected";
      }
      echo ">" . $option . "</option>";
    }
  ?>
</select><br>
<span class="error"><?php echo $spendenRhythmusErr;?></span>
<br>
```

Listing 3: Steuerelement für Spendenrhythmus in der Feldgruppe Spenden mit PHP erzeugen

Die Aufgabe der grau markierten Codezeilen in Listing 2 und Listing 3 werden im Abschnitt 4.3 (siehe Seite 10) erklärt.

4.2 Steuerelemente in der Feldgruppe Kreditkarte

Die fünf Steuerelemente (sog. HTML-Tags) in der Feldgruppe Kreditkarte des Spendenformulars sind:

1. Kartenmarke
2. Karteninhaber
3. Kartenummer
4. Ablaufdatum
5. Kartensicherheitscode

Diese 5 Steuerelemente werden mittels PHP-Code generiert, siehe Listing 4.

```
<fieldset>
  <legend>Kreditkarte</legend>
  <label for="kartenMarke">Kartenmarke<sup>*</sup></label>
  <select id="kartenMarke" name="kartenMarke[]" size="1" required="required">
    <option value="">---Bitte wählen---</option>
    <?php
      $marken = array("American Express", "Diners Int.", "MasterCard", "Visa");
      if (isset($_POST['kartenMarke'])){
        $kartenMarke = $_POST['kartenMarke'];
      } else {
        $kartenMarke = array();
      }
      foreach ($marken as $option){
        echo '<option value="' . $option . '"';
        if (in_array($option, $kartenMarke)){
          echo " selected";
        }
        echo ">" . $option . "</option>";
      }
    ?>
  </select>
  <span class="error"><?php echo $kartenMarkeErr; ?></span>
  <br>
  <label for "kartenInhaber">Karteninhaber<sup>*</sup></label>
  <input style="text-transform:uppercase;" type="text" name="kartenInhaber"
    size="30" required="required" placeholder="Karteninhaber"
    title="Bitte Namen eingeben genau wie auf der Kreditkarte angegeben."
    value="<?php echo isset($_POST['kartenInhaber']) ? htmlentities($_POST['kartenInhaber'])
: ''; ?>" />
  <span class="error"><?php echo $kartenInhaberErr;?></span>
  <br>
  <label for "kartenNummer">Kartenummer<sup>*</sup></label>
  <input type="text" name="kartenNummer" size="19" minlength="13" maxlength="16"
    required="required" placeholder="Kartenummer" pattern="[3|4|5|6].{12,}"
    title="Die erste Ziffer muss 3, 4, 5 oder 6 sein. Mindestlänge: 13 Zeichen."
    value="<?php echo isset($_POST['kartenNummer']) ? htmlentities($_POST['kartenNummer']) :
''; ?>" />
  <span class="error"><?php echo $kartenNummerErr;?></span>
  <br>
  <label for "kartenAblaufMM">Ablaufdatum<sup>*</sup></label>
  <select name="kartenAblaufMM" required="required">
    <?php
      $mm = "";
      if(isset($_POST['kartenAblaufMM'])){ $mm = $_POST['kartenAblaufMM'];}
      for ($m=1;$m<=12;++$m){
        if($m < 10){ $m = "0".$m;}
        echo '<option value="'.$m . '";'

```

```

        if ($m == $mm){echo " selected";}
        echo ">". $m."</option>";
    }
    ?>
</select>
<select name ="kartenAblaufJJ" required="required">
    <?php
        $jj = "";
        if (isset($_POST['kartenAblaufJJ'])){
            $jj = $_POST['kartenAblaufJJ'];
        }
        $jahr = date('Y');
        $jahrMax = $jahr + 5;
        for ($y=$jahr;$y<=$jahrMax;++$y){
            $jahr = substr($y, -2);
            echo '<option value='.$jahr;
            if ($jahr == $jj){
                echo " selected";
            }
            echo '>'.$jahr.'</option>';
        }
    ?>
</select>
<span class="error"><?php echo $kartenAblaufErr;?></span>
<br>
    <span style="padding-left:200px;padding-top:3px;">MM / JJ</span>
<br>
<label for="cvc">Sicherheitscode<sup>*</sup></label>
<input type="text" name="cvc" size="4" minlength="3" maxlength="4" required="required"
    pattern="[0-9]{3,4}" title="3 oder 4 Ziffern." placeholder="Nr."
    value="<?php echo isset($_POST['cvc']) ? htmlentities($_POST['cvc']) : ''; ?>" />
<span class="error"><?php echo $cvcErr;?></span>
</fieldset>

```

Listing 4: Steuerelemente in der Feldgruppe Kreditkarte

4.2.1 Kartenmarke

Die Auswahlliste für die *Kartenmarke* umfasst zurzeit 4 verbreitete Kartentypen, siehe Tab 4:

Kreditkarte			
Kurzname	Langname	Herausgebergruppe (MII*)	Länge Kartennummer
amex	AmericianExpress	3 (Reisen & Unterhaltung, Nichtbanken)	15
diners	Diners Club	3 (Reisen & Unterhaltung, Nichtbanken)	14
mc	MasterCard	5 (Banking & Finanzen)	16
visa	Visa	4 (Banking & Finanzen)	13 oder 16

*) Major Industry Identifier: Die erste Ziffer einer Kreditkartennummer verschlüsselt den MII.

Tab 4: Kreditkartenmarken und Länge der Kartennummer

4.2.2 Karteninhaber

Die Einbindung der CSS-Anweisung zur Formatierung des Karteninhabers erfolgt direkt im HTML-Code durch

```
style="text-transform:uppercase;
```

Damit wird erreicht, dass der Name des Karteninhabers im Spendenformular automatisch in Großbuchstaben umgewandelt wird.

4.2.3 Kartennummer

Das im `<input>`-Tag für *Kartennummer* verwendete *maxlength*-Attribut gibt die maximale Anzahl von Ziffern an, die zur korrekten Erfassung einer Kreditkartennummer benötigt werden. Maximal 19 Ziffern (ohne Gliederungszeichen) sind zulässig mit folgenden 3 Nummernteilen, vgl. [3]:

1. Nummernteil: Eine 6-stellige Verbundnummer des Kartenherausgebers, wobei deren erste Ziffer⁴ die Herausgebergruppe verschlüsselt, siehe Tab 4, Sp. 3.
2. Nummernteil: Eine Identnummer des jeweiligen Kontos mit einer variablen Länge von bis zu 12 Ziffern.
3. Nummernteil: Eine 1-stellige Prüfziffer für den sog. *Luhn-Algorithmus*⁵.

Mit dem *pattern*-Attribut wird die minimale Länge einer Kreditkartennummer sowie die zulässige Kennzahl der Herausgebergruppe vorgegeben.

4.2.4 Ablaufdatum

Das Ablaufdatum besitzt gewöhnlich das Format MM/JJ.

Der Monat MM des Ablaufdatums wird mit PHP als einfache Zählschleife von 1 bis 12 in Form einer Auswahlliste vorgegeben.

Beim Jahr JJ des Ablaufdatums muss sichergestellt werden, dass es nicht in der Vergangenheit liegt. Die zugehörige Auswahlliste wird erzeugt, indem in PHP mit $\$jahr = date('Y')$ das jeweils aktuelle Jahr die erste Option der Auswahlliste bildet und von dort aus drei Jahre hochgezählt wird. Dadurch aktualisiert sich die Auswahlliste für das Ablaufjahr des Spendenformulars automatisch.

4.2.5 Kartensicherheitscode

Die Eingabe des numerischen Kartensicherheitscodes ist Pflicht. Er umfasst meistens 3 Ziffern. Nur bei der Kartenmarke AmEx ist er 4 Ziffern lang. Nach dem Abschicken des vollständig ausgefüllten Spendenformulars wird dieser Sonderfall bei der Validierung der Eingaben mit PHP berücksichtigt. Die Plausibilisierung zwischen Kartenmarke und Kartensicherheitscode kann in PHP mit einer IF-ELSE-Verzweigung durchgeführt werden, siehe Listing 5. Abb. 2 beinhaltet die entsprechende Fehlermeldung in der Feldgruppe *Kreditkarte* des Spendenformulars.

⁴ Kennzahl der Hauptbranche: **Major Industry Identifier**, Kürzel: MII

⁵ Modulo 10 Prüfzifferberechnung

```

if (empty($_POST["cvc"])) {
    $cvcErr = "Pflichteingabe.";
    $hasErrors = true;
} else {
    $cvc = test_input($_POST["cvc"]);
    if ($kartenMarke == "American Express") {
        if (strlen($cvc) != 4) {
            $cvcErr = "4-stelliger Kartensicherheitscode erforderlich!";
            $hasErrors = true;
        }
    } else {
        if (strlen($cvc) != 3) {
            $cvcErr = "3-stelliger Kartensicherheitscode erforderlich!";
            $hasErrors = true;
        }
    }
}
}
}

```

Listing 5 : Plausibilitätskontrolle zwischen Kartenmarke und Kartensicherheitscode (cvc)

Kreditkarte

Kartenmarke * :

Karteninhaber * :

Kartennummer * :

Ablaufdatum * :
MM / JJ

Sicherheitscode * : **4-stelliger Kartensicherheitscode erforderlich!**

Abb. 2: Fehlermeldung (rot) bei mangelnder Plausibilität zwischen Kartenmarke und Kartensicherheitscode

4.3 Erfasste Formulardaten nach der Validierung erneut anzeigen

Für den Benutzer des Spendenformulars wäre ärgerlich, wenn vor dem Abschicken des Formulars erfasste Eingabedaten bei der Validierung verloren gehen würden und deshalb erneut eingegeben werden müssten, falls Fehler gefunden werden. Um bereits erfasste Formulardaten zu erhalten, wird eine Technik eingesetzt, die in [4] an einem praktischen Beispiel demonstriert wird. Die Code-Zeile

```
value="<?php echo isset($_POST['cvc']) ? htmlentities($_POST['cvc']) : ''; ?>" />
```

belegt beispielsweise das Textfeld für den *Kartensicherheitscode* (Name: *cvc*) erneut mit dem zuvor erfassten Wert, vgl. Listing 4. Bei fehlerhaften Formulardaten bewirkt die Code-Zeile

```
<span class="error"><?php echo $cvcErr;?></span>
```

die Anzeige einer roten Feldmeldung rechts neben dem zugehörigen Textfeld, siehe Abb. 2.

Für Auswahllisten ist die Wiederbelegung⁶ der Formularfelder mit den Namen *spendenBetrag* (vgl. Listing 2) und *spendenRhythums* (vgl. Listing 3) etwas aufwändiger. Die entsprechenden PHP-Codezeilen sind dort grau unterlegt, um die Programmierlogik hervorzuheben.

5 Plausibilitätskontrolle des Spendenformulars

Alle Feldgruppen des Spendenformulars werden nach dem Abschicken rigoros auf Plausibilität geprüft, soweit das nicht bereits mit Attributen der dort verwendeten HTML-Tags geschehen ist. Zwei Beispiele werden im Folgenden erörtert.

5.1 Plausibilität zwischen Land und Postleitzahl

In Deutschland ist die Postleitzahl 5-stellig, in Österreich und der in Schweiz 4-stellig. Die entsprechende Plausibilitätskontrolle erfolgt mit folgendem PHP-Code, siehe Listing 6 und dazugehöriger Abb. 3.

```
switch ($land) {
    case "at":
        If(strlen($plz) !=4) {
            $plzErr = "Länge falsch!"; $hasErrors = true;
        }
    case "ch":
        If(strlen($plz) !=4) {
            $plzErr = "Länge falsch!"; $hasErrors = true;
        }
        break;
    case "d":
        IF (strlen($plz) != 5) {
            $plzErr = "Länge falsch!"; $hasErrors = true;
        }
        break;
    default:
        break;
}
```

Listing 6: Plausibilitätskontrolle zwischen Land und Länge der Postleitzahl (plz)

The screenshot shows a form titled "Spender" with the following fields and error messages:

- Anrede ***: Radio buttons for "Frau" and "Herr". "Herr" is selected.
- Vorname ***: Input field with "hans", error: "Initiale in Kleinschreibung!"
- Nachname ***: Input field with "müller", error: "Initiale in Kleinschreibung!"
- Ihr Foto:**: Input field with "Durchsuchen..." button.
- Str. mit HausNr. ***: Input field with "Ulmenstraße", error: "Hausnummer fehlt!"
- Wohnort ***: Input field with "Ratingen".
- Postleitzahl ***: Input field with "4088", error: "Länge falsch!"
- Land ***: Dropdown menu with "Deutschland" selected.

Abb. 3: Fehlermeldungen (rot) bei mangelnder Plausibilität der Fomulardaten

⁶ (*repopulate | refresh | redisplay | reload | keep*) form data after (*submit | submission | validation*) sind Erfolg versprechende Suchwörter im Internet für die Wiederbelegung von Formularfeldern nach dem Abschicken des jeweiligen Formulars.

5.2 Existenzkontrolle der Hausnummer im Textfeld 'strasse'

Das Spendenformular umfasst in der Feldgruppe *Spender* kein separates Eingabefeld für die Hausnummer in einer Straße. Sie soll auch im Textfeld für Straße erfasst werden. Wenn dort die Existenz einer Hausnummer nicht festgestellt werden kann, soll im Spendenformular die Fehlermeldung „Hausnummer fehlt!“ angezeigt werden, siehe Abb. 3.

Die Existenz einer Hausnummer im Textfeld 'strasse' kann mit der PHP-Funktion *Hausnummer_finden* durchgeführt werden, siehe Listing 7, linke Spalte. Der reguläre Ausdruck in der *preg_match* Anweisung wurde aus [4] übernommen.

PHP-Skript:	Ergebnisse:
<pre><?php // Funktion Hausnummer_finden: // Gibt zurück, ob im Textfeld 'strasse' eine // Hausnummer enthalten ist. function Hausnummer_finden(\$strasse) { \$match = array(); preg_match('/^[^\d]*[^\d\s] *(\d.*)\$/',\$strasse,\$match); if(count(\$match) < 2) { return false; } else { return true; } } \$strasse = array("Weinbergweg 12", "Weinbergweg 2a", "Weinbergwegl2a", "Untere Steinhauser Str. 10", "10 Untere Steinhauser Str.", "101Untere Steinhauser Str.", "Ulmenstraße"); foreach (\$strasse as \$str => \$nm) { \$hatHausnummer = Hausnummer_finden(\$nm); \$msg = \$nm." enthält "; if (\$hatHausnummer){ echo \$msg." eine Hausnummer "
"; } else { echo \$msg." keine Hausnummer "
"; } } ?></pre>	<pre>'Weinbergweg 12' enthält eine Hausnummer 'Weinbergweg 2a' enthält eine Hausnummer 'Weinbergwegl2a' enthält eine Hausnummer 'Untere Steinhauser Str. 10' enthält eine Hausnummer '10 Untere Steinhauser Str.' enthält eine Hausnummer '101Untere Steinhauser Str.' enthält eine Hausnummer 'Ulmenstraße' enthält keine Hausnummer</pre>

Listing 7: Hausnummer im Textfeld 'strasse' finden

5.3 Gültigkeit der Eingaben in die Textfelder für Vor- und Nachname

Mit der PHP-Funktion `parse_name` kann ein Textfeld auf Gültigkeit geprüft werden, das einen Namen enthält:

- Anfangsbuchstabe in Großschreibung
- Umlaute dürfen im Namen vorkommen
- Punkt, Bindestrich und Leerzeichen sind zugelassen
- Ziffern dürfen im Namen *nicht* vorkommen.

<p>PHP-Skript:</p> <pre><?php // Funktion parse_name (Namen prüfen): // Anfangsbuchstabe in Großschreibung, Punkte, // Leerzeichen und Bindestriche sind zugelassen. // Ziffern dürfen im Namen nicht vorkommen. function parse_Name(\$nm) { \$pattern = "/^[A-ZÄÖÜ]{1}[A-Za-züäöÄÖÜß\.\-]{2,}\$/"; if (preg_match(\$pattern, \$nm)) { return "richtig"; } else { return "falsch"; } } } \$names = array('Hans-Peter Hansen', 'H.-P. Hansen', 'h. P. Hansen', 'H. P. Hansen', 'H. P. Hansen--Schulte', 'Hansl - Peter Hansen-Schulte', 'e. Fente-Schmidt', 'Ella Fente-Schmidt', 'Fritz Köhnen', 'Öztürk Yilmaz', 'Heinz Nußbaum',); foreach (\$names as \$nm) { \$result = parse_name(\$nm); echo \$nm."": ".\$result."
"; } ?></pre>	<p>Ergebnisse:</p> <table><tr><td>Hans-Peter Hansen:</td><td>richtig</td></tr><tr><td>H.-P. Hansen:</td><td>richtig</td></tr><tr><td>h. P. Hansen:</td><td>falsch</td></tr><tr><td>H. P. Hansen:</td><td>falsch</td></tr><tr><td>H. P. Hansen--Schulte:</td><td>richtig</td></tr><tr><td>Hansl - Peter Hansen-Schulte:</td><td>falsch</td></tr><tr><td>e. Fente-Schmidt:</td><td>falsch</td></tr><tr><td>Ella Fente-Schmidt:</td><td>richtig</td></tr><tr><td>Fritz Köhnen:</td><td>richtig</td></tr><tr><td>Öztürk Yilmaz:</td><td>richtig</td></tr><tr><td>Heinz Nußbaum:</td><td>richtig</td></tr></table>	Hans-Peter Hansen:	richtig	H.-P. Hansen:	richtig	h. P. Hansen:	falsch	H. P. Hansen:	falsch	H. P. Hansen--Schulte:	richtig	Hansl - Peter Hansen-Schulte:	falsch	e. Fente-Schmidt:	falsch	Ella Fente-Schmidt:	richtig	Fritz Köhnen:	richtig	Öztürk Yilmaz:	richtig	Heinz Nußbaum:	richtig
Hans-Peter Hansen:	richtig																						
H.-P. Hansen:	richtig																						
h. P. Hansen:	falsch																						
H. P. Hansen:	falsch																						
H. P. Hansen--Schulte:	richtig																						
Hansl - Peter Hansen-Schulte:	falsch																						
e. Fente-Schmidt:	falsch																						
Ella Fente-Schmidt:	richtig																						
Fritz Köhnen:	richtig																						
Öztürk Yilmaz:	richtig																						
Heinz Nußbaum:	richtig																						

Abb. 4: PHP-Skript zur Validierung von Namen

5.4 Gültigkeitsprüfung einer Kreditkarte

Die Plausibilitätskontrolle einer Kreditkarte erfolgt in 7 Schritten mit der PHP-Funktion `checkCreditCard`, der sowohl die Kartenummer als auch die Kartenmarke als Argumente übergeben werden können.

1. Prüfen, ob eine Kartenummer übergeben wurde. Wenn ja, Kartenummer bereinigen: Alle nicht-nummerischen Zeichen entfernen.
2. Wenn keine Kartenmarke übergeben wurde, diese anhand der Kartenummer und deren Muster (engl. *pattern*) bestimmen.
3. Wenn die übergebene Kartenmarke nicht übereinstimmt mit einer gespeicherten im 2-dimensionalen Datenvektor `$Karten`, einen Fehler melden.
4. Die erste Ziffer der Kartenummer prüfen, vgl. Tab 4, Sp. 3.
5. Prüfung der Kartenummer auf gültiges Muster mit entsprechender PHP `preg_match`-Anweisung.

6. Länge der Kartennummer prüfen in Abhängigkeit von der Kartenmarke.
7. Modulo-10-Prüfziffer der Kartennummer berechnen.

Wenn alle 7 Prüfungen positiv ausfallen, ist die Kreditkarte mit ziemlicher Sicherheit gültig, siehe Listing 8. Die Prüfungen beruhen größtenteils auf Metadaten, die später in Listing 11 dargestellt werden.

```
<?php
/*
Diese Routine überprüft die Kartennummer der jeweiligen Kreditkartenmarke.
In Anlehnung an: John Gardner, Auto credit check?, 4. Jan. 2005
https://www.webmasterworld.com/php/3338218.htm
Die Nummernteile der Kartenmarken wurden aus einer Vielzahl von Quellen zusammengetragen.
Die beste Quelle dafür ist wahrscheinlich auf Wikipedia ("Payment card number"):
https://en.wikipedia.org/wiki/Payment_card_number
Reguläre Ausdrücke zur Überprüfung von Kreditkartennummern sind u. a. zu finden in:
'Validating Credit Card Numbers on Your Order Form':
https://www.regular-expressions.info/creditcard.html
=====
*/
function checkCreditCard ($kartenNummer, $kartenMarke) {

    include 'defineCreditCards.php'; // Definition der Metadaten von Kreditkarten.

    // Datenfeld für Fehlermeldungen
    $fehlerMld[0] = "Unbekannte Kartenmarke.";
    $fehlerMld[1] = "Keine Kreditkartennummer angegeben.";
    $fehlerMld[2] = "Kreditkartennummer hat ungültiges Format.";
    $fehlerMld[3] = "Kreditkartennummer ist ungültig.";
    $fehlerMld[4] = "Kreditkartennummer hat falsche Länge.";

    /* 1. Prüfen, ob eine Kartennummer übergeben wurde.*/
    if (strlen($kartenNummer)==0) {
        $fehlerNum = '1';
        $fehlerText = $fehlerMld[$fehlerNum];
        return false;
        // Wenn ja, Kartennummer bereinigen.
    } else {
        // Nicht-nummerische Zeichen aus der Kreditkartennummer entfernen.
        $kartenNummer = preg_replace('/[^\\d]/', '', $kartenNummer);
    }
}
```

```
/* 2. Wenn keine Kartenmarke übergeben wurde,
diese anhand der Kartennummer und deren Muster (engl. pattern) bestimmen: */
if (strlen($kartenMarke)==0) {
    $match = false;
    foreach ($karten as $marke => $merkmale) {
        $muster = $merkmale['muster'];
        if (strlen($muster) > 0 && preg_match($muster,$kartenNummer)){
            $match=true;
            $kartenKey = $marke;
            break;
        }
    }
}
if ($match == false) {
    $fehlerNum = '0';
}
```



```

    $fehlerText = $fehlerMld[$fehlerNum];
    return false;
}
/* Wenn eine Kartennummer wurde übergeben,
damit den Schlüssel bestimmen. */
} else {
    $kartenKeyOK = false;
    foreach ($karten as $marke => $merkmale) {
        $brand = strtolower($merkmale['marke']);
        if ($brand == strtolower($kartenMarke)) {
            $kartenKey = $marke;
            $kartenKeyOK = true;
            break;
        }
    }
    if ($kartenKeyOK == false) {
        $fehlerNum = '0';
        $fehlerText = $fehlerMld[$fehlerNum];
        return false;
    }
}
/* 3. Wenn die übergebene Kartenmarke nicht übereinstimmt mit einer gespeicherten
im 2-dimensioanlen Datenvektor (d. h. Metadaten), eine Fehlermeldug ausgeben:*/
$markerOK = false;
foreach($karten as $marke => $merkmale) {
    if (strtolower($marke) == strtolower($kartenKey)) {
        $markerOK = true;
        break;
    }
}
if ($markerOK == false) {
    $fehlerNum = '0';
    $fehlerText = $fehlerMld[$fehlerNum];
    return false;
}
}

```

```

// 4. Erste Ziffer der Kartennummer prüfen
$firstDigitOK = false;
$ziffer1 = substr($kartenNummer, 0, 1);
$lengths = split(',', $karten[$kartenKey]['mii']);
for ($j=0;$j<=sizeof($lengths);$j++) {
    if ($ziffer1 == $lengths[$j]) {
        $firstDigitOK = true;
        break;
    }
}
if (!$firstDigitOK) {
    $fehlerNum = '3';
    $fehlerText = $fehlerMld [$fehlerNum];
    return false;
};

// 5. Prüfung der Kartennummer auf gültiges Muster
$musterOK = false;
$muster = $karten[$kartenKey]['muster'];
if (preg_match($muster, $kartenNummer)) {
    $musterOK = true;
}
if (!$musterOK) {
    $fehlerNum = '5';
    $fehlerText = $fehlerMld [$fehlerNum];
    return false;
}

// 6. Länge der Kartennummer prüfen
$laengeOK = false;
$lengths = split(',', $karten[$kartenKey]['laenge']);
for ($j=0;$j<=sizeof($lengths);$j++) {
    if(strlen ($kartenNummer )==$lengths[$j]) {
        $laengeOK = true;
        break;
    }
}
if (!$laengeOK) {
    $fehlerNum = '4';
    $fehlerText = $fehlerMld [$fehlerNum];
    return false;
};

```

```

// 7. Prüfziffer berechnen, falls erforderlich.
if ($karten[$kartenKey]['mod10']) {
    $pruefSumme = 0; // laufende Prüfsumme
    $ziffer = ""; // nächste zu bearbeitende Ziffer
    $j = 1; // Die Variable j nimmt alternierend die Werte 1 oder 2 an.
    // Jedes Ziffer einzeln verarbeiten, rechts beginnend.
    for ($i = strlen($kartenNummer) - 1; $i >= 0; $i--) {
        // Die nächste Ziffer extrahieren und diese alternierend mit 1 oder 2 multiplizieren.
        $ergebnis = $kartenNummer{$i} * $j;
        // Wenn das Ergebnis zweistellig ist, addiere 1 zur Prüfsumme.
        if ($ergebnis > 9) {
            $pruefSumme = $pruefSumme + 1;
            $ergebnis = $ergebnis - 10;
        }
        // Prüfsumme erhöhen.
        $pruefSumme = $pruefSumme + $ergebnis;
        // Den Wert der Variablen j umschalten.
        if ($j == 1) {$j = 2;} else {$j = 1;};
    }
    // Wenn die Prüfsumme ohne Rest durch 10 teilbar ist, liegt ein richtiges Ergebnis vor.
    // Wenn nicht, einen Fehler melden.
    if ($pruefSumme % 10 != 0) {
        $fehlerNum = '3';
        $fehlerText = $fehlerMld [$fehlerNum];
        return false;
    }
}
// Die Kreditkarte besitzt das passende Format.
return true;
}

```

Listing 8: PHP-Code zur Validierung einer Kreditkarte mittels Metadaten

5.5 Testdaten für die PHP-Funktion 'checkCreditCard'

Im zweidimensionalen assoziativen Array `$cc` sind je Kreditmarke (amex, diners, mc und visa) etliche Kreditkartennummern gespeichert. Ihre Gültigkeit wird mit der PHP-Funktion `checkCreditCard` überprüft.

```
// Zu überprüfende Kreditkartennummern von 4 Kreditkartenmarken
$cc = array(
    "amex" => array('378282246310005', '371449635398431', '34343434343434', '379762088600299'),
    "diners" => array('30569309025904', '38520000023237', '36700102000000', '36148900647913'),
    "mc" => array('5555555555554444', '5105105105105100', '5454545454545454', '2221000000000009',
        '2222420000001113', '2222630000001125', '5238801032609491'),
    "visa" => array('4012888888881881', '4444333322221111', '49118300000000', '4917610000000000',
        '4111111111111111', '4012888888881881', '4887704455185832')
);

// Gültigkeit vorstehender Kreditkartennummern prüfen
foreach ($cc as $kartenMarke => $arr) {
    echo "Kartenmarke: " . $kartenMarke . "<br>";
    foreach ($arr as $key => $kartenNummer) {
        if (checkCreditCard($kartenNummer, $kartenMarke)) {
            echo 'Die Kartennummer ' . $kartenNummer . ' ist gültig.' . "<br>";
        } else {
            echo 'Die Kartennummer ' . $kartenNummer . ' ist ungültig.' . "<br>";
        }
    }
    echo '<br>';
}
?>
```

Listing 9: Test-Kreditkartennummern bekannter Kreditkartenmarken (Quelle: [5], [6])

Die Ergebnisse der Gültigkeitsprüfung sind in Listing 10 aufgeführt:

Kartenmarke: **amex**

Die Kartennummer 378282246310005 ist gültig.

Die Kartennummer 371449635398431 ist gültig.

Die Kartennummer 34343434343434 ist ungültig⁷.

Die Kartennummer 379762088600299 ist gültig.

Kartenmarke: **diners**

Die Kartennummer 30569309025904 ist gültig.

Die Kartennummer 38520000023237 ist gültig.

Die Kartennummer 36700102000000 ist gültig.

Die Kartennummer 36148900647913 ist gültig.

Kartenmarke: **mc**

Die Kartennummer 5555555555554444 ist gültig.

Die Kartennummer 5105105105105100 ist gültig.

Die Kartennummer 5454545454545454 ist gültig.

Die Kartennummer 2221000000000009 ist gültig.

Die Kartennummer 2222420000001113 ist gültig.

Die Kartennummer 2222630000001125 ist gültig.

Die Kartennummer 5238801032609491 ist gültig.

Kartenmarke: **visa**

Die Kartennummer 4012888888881881 ist gültig.

Die Kartennummer 4444333322221111 ist gültig.

Die Kartennummer 49118300000000 ist gültig.

Die Kartennummer 4917610000000000 ist gültig.

Die Kartennummer 4111111111111111 ist gültig.

Die Kartennummer 4012888888881881 ist gültig.

Die Kartennummer 4887704455185832 ist gültig.

Listing 10: Überprüfte Kreditkartennummern von vier bekannten Kreditkartenmarken

⁷ American Express (**amex**) Kreditkartennummern müssen 15 Zeichen lang sein. Es sind aber nur 14 Zeichen.

5.6 Metadaten für die Gültigkeitsprüfung von Kreditkarten

Um generische Kartentypen für die Gültigkeitsprüfung von Kreditkarten zu verwenden, wird folgende Datenstruktur in Anlehnung an [7] herangezogen:

```
<?php
/*
Metadaten für ausgewählte Kreditkartenmarken definieren.
Weitere Kartenmarken können mit folgenden sechs Merkmalen hinzugefügt werden:
marke: 1. Marke (engl. 'brand') der Kreditkarte
muster: 2. Muster (engl. 'pattern') zur Formatprüfung der Kreditkartennummer
        mittels regulärem Ausdruck (engl. 'regular expression').
laenge: 3. Länge der Kreditkartennummer, wobei 12 <= laenge <= 19.
mii: 4. Erste Ziffer (engl. 'major industry identifier') der Kreditkartennummer.
kpn: 5. Länge der Kartenprüfnummer (engl. 'card verification value')
mod10: 6. Wenn 'true', ist eine Prüfziffernberechnung nach Modulus 10 erforderlich.
*/
$karten = array(
    "amex" => array(
        'marke' => 'American Express',
        'muster' => '/^([34|37]{2}) ([0-9]{13})$/ ',
        'laenge' => '15',
        'mii' => '3',
        'kpn' => '4',
        'mod10' => true,
    ),
    "diners" => array(
        'marke' => 'Diners Int.',
        'muster' => '/^([30|36|38]{2}) ([0-9]{12})$/ ',
        'laenge' => '14',
        'mii' => '3',
        'kpn' => '3',
        'mod10' => true,
    ),
    "mc" => array(
        'marke' => 'MasterCard',
        'muster' => '/^5[1-5]\d{14}$|^2(?:2(?:2[1-9]| [3-9]\d)| [3-6]\d\d|7(?:[01]\d|20))\d{12}$/ ',
        'laenge' => '16',
        'mii' => '5,2',
        'kpn' => '3',
        'mod10' => true,
    ),
    "visa" => array(
        'marke' => 'Visa',
        'muster' => '/^([4]{1}) ([0-9]{12,15})$/ ',
        'laenge' => '13,16,19',
        'mii' => '4',
        'kpn' => '3',
        'mod10' => true,
    )
);
?>
```

Listing 11: Metadaten für die Gültigkeitsprüfung von Kreditkarten ('defCreditCards.php')

Beim Aufruf der PHP-Funktion *checkCreditCard* wird dort die Datei *defineCreditCards.php* mittels *include*-Anweisung eingebunden.

6 Auswertung der Daten des Spendenformulars

In [1] und [9] wird gezeigt, wie die Daten eines HTML-Spendenformulars für die Erstellung einer Spendenbescheinigung genutzt werden können. Deshalb wird hier nicht weiter dargestellt, für welche Zwecke die Daten eines Spendenformulars ausgewertet werden können.

7 Literatur- und Quellenverzeichnis

- [1] U. Günther, „PHP 5 - Ein praktischer Einstieg,“ O'Reilly, 08 2004. [Online]. Available: <https://www.oreilly.de/german/freebooks/einphp2ger/ch04.html>. [Zugriff am 23 05 2018].
- [2] E. Wetsch, Einstieg in CSS, Grundlagen und Praxis, 2. Aufl. Hrsg., Bonn: Galileo Press, 2011, p. 438.
- [3] o. V., „Payment card number,“ Wikipedia, 9 06 2018. [Online]. Available: https://en.wikipedia.org/wiki/Payment_card_number. [Zugriff am 10 06 2018].
- [4] o. V., „PHP Form Validation Example,“ w3schools.com, [Online]. Available: https://www.w3schools.com/php/showphp.asp?filename=demo_form_validation_complete. [Zugriff am 13 06 2018].
- [5] R. Tobias, „PHP: Straßenname und Hausnummer mit PHP parsen,“ 25 10 2014. [Online]. Available: <https://www.tricd.de/php/php-strassenname-und-hausnummer-mit-php-parsen/>. [Zugriff am 07 06 2018].
- [6] o. V., „Test Credit Card Account Numbers,“ o. J. [Online]. Available: https://www.paypalobjects.com/en_AU/vhelp/paypalmanager_help/credit_card_numbers.htm. [Zugriff am 10 06 2018].
- [7] o. V., „Test and Go Live Guide, Test card numbers,“ worldpay, o.J. [Online]. Available: <http://support.worldpay.com/support/kb/bg/testandgolive/tg!5103.html>. [Zugriff am 10 06 2018].
- [8] J. Gardner, „FormBuilder-Development / php / phpcreditcard.php,“ 2002 02 2012. [Online]. Available: <https://github.com/TheLifeProject/FormBuilder-Development/blob/master/php/phpcreditcard.php>. [Zugriff am 10 06 2018].
- [9] it-heberle, „Problem mit getdate,“ lima-city, 27 01 2011. [Online]. Available: <https://www.lima-city.de/thread/hilfe-php>. [Zugriff am 10 06 2018].
- [10] o. V., „Validating Credit Card Numbers on Your Order Form,“ Regular-Expressions, 21 09 2017. [Online]. Available: <https://www.regular-expressions.info/creditcard.html>. [Zugriff am 10 06 2018].
- [11] S. Münz und C. Gull, HTML5 Handbuch, 9. Auflage Hrsg., Haar bei München: Franzis Verlag, 2013, p. 779.