PTraffic-Library
Version 0.8

Handbuch

Inhaltsverzeichnis

1. Einführung

PTraffic-Library ist eine Javascript-Bibliothek für die Programme PTraffic (Pro/ProPlus) und Linemap Draw. Mit der Bibliothek ist es mögliche einfach und schnell Fahrplananwendungen für das Internet zu entwickeln.

1.1. Konventionen

Wenn in dieser Dokumentation der Name PTraffic verwendet wird sind damit, wenn nicht anders angegeben, die Programme PTraffic, PTraffic Pro, PTraffic ProPlus sowie LineMap Draw (Home bis XL) ebenfalls mitgemeint.

1.2. Download

Die benötigten Dateien können hier heruntergeladen werden:

PTraffic-Bibliothek: www.de.ptraffic.net/ptraffic-library/index.htm

PTraffic-Pro Datenmodell (PDF)::www.ptraffic.net/PTraffic_Pro_Datenmodell_1_0.pdf

1.3. Arbeiten mit der PTraffic-Library

Die PTraffic-Library baut auf der Javascript-Bibliothek PublicSQL auf. Diese Bibliothek ermöglicht einfache Datenbank-Abfragen in Javascript. PublicSQL arbeitet mit dem in PTraffic verwendeten Datenformat so dass auf sämtliche Fahrplandaten mit PublicSQL zugegriffen werden kann.

Link zur PublicSQL-Website: www.publicsql.org

2. Schnelleinstieg

In diesen Tutorial für PTraffic und PTraffic Pro/ProPlus wird ein Fahrplananzeige mit der zugehörigen Eingabemaske erstellt.

Erstellen Sie zunächst ein neues Verzeichnis und darin die Unterverzeichnisse "ptf" und "javascript". Anschließend kopieren Sie die Projektdateien Ihres PTraffic-Projekts, also alle Dateien mit der Datei-Endung ".ptf" und die Projektdatei mit der Eindung ".ppr", in das Vezeichnis "ptf". Dann kopieren Sie noch die Javascript-Dateien aus der PTraffic-Library in das Verzeichnis "javascript".

Erstellen Sie nun das folgende HTML-Grundgerüst:

```
<!DOCTYPE html>
<html lang="de">
  <head>
    <meta charset="iso-8859-1" />
    <script src="javascript/ptraffic_text.js"
type="text/javascript"></script>
    <script src="javascript/ptraffic.js"
type="text/javascript"></script>
    <script src="javascript/ptraffic_timetable.js"
type="text/javascript"></script>
    <script src="javascript/publicsql.js"
type="text/javascript"></script>
    <title>PTraffic-Library Schnelleinstieg(1)</title>
  </head>
  <body>
  </body>
</html
```

Im ersten <meta>-Tag wird der Zeichensatz "iso-8859-1" angegeben, da PTraffic in der aktuellen Version kein Unicode-Zeichensatz unterstützt. Die HTML-Datei darf daher auch nicht als Unicode-Datei gespeichert werden sond ern sollte als ASCII- bzw. ANSI-Datei gespeichert werden, sonst werden Umlaute und Sonderzeichen nicht korrekt dargestellt.
Anschließend werden alle benötigten Javascript-Dateien eingebunden.
Zwischen <body...> und </body> geben Sie nun den folgenden Quelltext ein:

```
<div id="mydiv">
</div>

<table id="mytable" border="2">
</table>

<script>

function show() {
  showTimetableForm("mydiv", "mytable");
}

ptInit(withTimetable, 'ger', 'ptf', show);

</script>
```

Im -Container soll später die Eingabemaske erscheinen. Die Tabelle ist für die Fahrplananzeige. Hier geben wir noch einen Rahmen mit 2 Pixel Breite an ("border="2").

Mit ptInit() wird die PTraffic-Library initialisiert.
Der Parameter "withTimetable" steht für die verwendete Programmversion (hier PTraffic, PTraffic Pro/ProPlus). Als Sprache wird hier deutsch ("ger") gewählt und als Verzeichnis für die

Projektdateien das Verzeichnis "ptf". Zusätzlich wird als Callback-Funktion die Funktion "show" übergeben. Dies bedeutet, dass die Funktion "show" nach der Initialisierung aufgerufen wird.

In der Funktion "show" wird die Funktion "showTimetableForm" aufgerufen. Diese zeigt eine Eingabemaske zur Auswahl eines Fahrplans an.
Als Parameter wird die id des -Tags für die Eingabemaske und die id der Tabelle für den Fahrplan angegeben.

Anzeigebeispiel - so sieht's aus: www.ptraffic.net/ptraffic-library/Beispiele/Schnelleinstieg1.htm

2.1. Weitere Beispiele

Auf die gleiche Weise können wir viele weitere Funktionen aufrufen. Dazu muss nur der Funktionsname "showTimetable" ausgetauscht werden - die Parameter sind identisch:

showLineCourseForm (Linienverlauf)
showCurrentRidesForm (Fahrplan mit aktuell stattfindende Fahrten anzeigen)
showTimetablePeriodForm (Fahrplan für einen bestimmten Zeitraum anzeigen)
showStationTimetableForm (Stations-Fahrplan)
showDeparturesForm (Stations-Abfahrten)

Für die Anzeige von Bildfahrplänen wird statt des <table>-Elements ein <canvas>-Element benötigt:

```
<canvas id="mycanvas" style="border:2px">
</canvas>
```

hier wurde noch ein schwarzer Rahmen mit 2 Pixel Breite angegeben. Der Funktionsaufruf sieht dann so aus:

```
showGraphicalTimetableForm("mydiv", "mycanvas");
```

Anzeigebeispiel - so sieht's aus: www.ptraffic.net/ptraffic-library/Beispiele/Schnelleinstieg2.htm

Für die Anzeige aller Stationen oder aller Linien können wir das -Element löschen. Als Parameter wird nur die id des <table>-Elements übergeben:

```
showStations("mytable");
showLines("mytable");
```

2.2. Alle Beispiele in einer Anwendung zusammenfassen

Wie wir gesehen haben werden alle Funktionen ähnlich aufgerufen. Unterschiedlich ist lediglich, welche der HTML-Elemente von den 3 Elementen , <table> oder <canvas> wir als Parameter übergeben. Es ist daher einfach, eine Anwendung zu schreiben in der wir aus den vorhandenen Funktionen die passende auswählen.

Dazu erstellen wir zunächst die passende Auswahlbox:

```
Auswahl:
<select name="sbAuswahl" id="sbAuswahl" onChange="show();">
  <option selected value="1">Stationen</option>
  <option value="2">Linien</option>
  <option value="3">Linienverlauf</option>
  <option value="4">Fahrplan</option>
  <option value="5">aktuelle Fahrten</option>
  <option value="6">Zeitraum-Fahrplan</option>
  <option value="7">Stations-Fahrplan</option>
  <option value="8">Stations-Abfahrten</option>
  <option value="9">Bildfahrplan</option>
</select>
```

```
    <br>
```

Die Werte "1" bis "9" werden später benutzt um die richtige Funktion für das gewählte Element aufzurufen. Der Parameter "selected" bei der ersten Option bewirkt dass diese vorausgewählt wird. Der onChange-Parameter legt fest dass bei einer Änderung der Auswahl die Funktion show() aufgerufen wird.

Anschließend legen wir die 3 HTML-Elemente (, <table> und <canvas>) an:

```
    <div id="mydiv">
    </div>

    <table id="mytable" border="2">
    </table>

    <canvas id="mycanvas" style="border:2px solid #000000;">
    </canvas>
```

Nun folgt der Javascript-Abschnitt mit der Funktion show() um die ausgewählte Funktion aufzurufen:

```
<script>

function show() {
  var wert =
document.getElementById("sbAuswahl").options[document.getElementById("sbAuswahl"
).selectedIndex].value;
  var cv = document.getElementById("mycanvas");
  var context = cv.getContext('2d');
  document.getElementById("mydiv").innerHTML = "";
  document.getElementById("mytable").innerHTML = "";
  document.getElementById("mycanvas").style.visibility = (wert == 9) ? "visible"
: "hidden";
  cv.height = 200;
  cv.width = 200;
  context.fillStyle = "white";
  context.fillRect(0, 0, cv.width, cv.height);
  switch (wert)   {
    case "1":
      showStations("mytable");
      break;
    case "2":
      showLines("mytable");
      break;
    case "3":
      showLineCourseForm("mydiv", "mytable");
      break;
    case "4":
      showTimetableForm("mydiv", "mytable");
      break;
    case "5":
      showCurrentRidesForm("mydiv", "mytable");
      break;
    case "6":
      showTimetablePeriodForm("mydiv", "mytable");
      break;
    case "7":
      showStationTimetableForm("mydiv", "mytable");
      break;
    case "8":
      showDeparturesForm("mydiv", "mytable");
      break;
    case "9":
```

```
      showGraphicalTimetableForm("mydiv", "mycanvas");
      break;
  }
}
```

In "wert" wird der Wert (value) für die ausgewählte Funktion gespeichert. die Variablen "cv" und "context" werden für den Zugriff auf das Canvas-Element benötigt.
Anschließend werden die Inhalte der div- und table-Elemente geleert. Das canvas-Element wird abhängig von der gewählten Funktion unsichtbar oder sichtbar (bei Bildfahrplan-Funktion) geschaltet. Außerdem wird der Canvas-Inhalt gelöscht (mit weißer Hintergrundfarbe überschrieben) und die Größe auf 200x100 Pixel geändert.

In der case-Anweisung wird anschließend abhängig vom Wert der Auswahlbox die richtige Funktion aufgerufen.

Zum Schluß wird wie zuvor die PTraffic-Library initialisiert und der Javascipt-Teil beendet.:

```
ptInit(withTimetable, 'ger', 'ptf', show);

</script>
```

Durch die Übergabe von "show" als Callback-Funktion wird für die vorselektierte Auswahl "Stationen" die zugehörige Funktion aufgerufen.

Anzeigebeispiel - so sieht's aus: www.ptraffic.net/ptraffic-library/Beispiele/Schnelleinstieg3.htm

2.2.1. Kleine Verbesserungen

Teilweise erscheint bei einigen Browsern bei der Auswahl des Bildfahrplans ein kleiner Punkt an der Position des <table>-Elements. Anstatt nur das <canvas>-Element abhängig von der gewählten Funktion unsichtbar oder sichtbar zu schalten können wir dies auch mit dem <table>-Element machen. Dazu fügen wir in der Funktion "show" folgende Zeile ein (rot):

```
...
document.getElementById("mycanvas").style.visibility = (wert == 9) ? "visible" :
"hidden";
document.getElementById("mytable").style.visibility = (wert != 9) ? "visible" :
"hidden";
cv.height = 200;
...
```

Außerdem wollen wir erreichen dass beim Aufruf der Website zunächst zunächst nichts ausgewählt ist. Ersetzen Sie die Zeile mit dem Aufruf von "ptInit" durch folgenden Quelltext:

```
document.getElementById("sbAuswahl").selectedIndex = -1;
document.getElementById("mycanvas").style.visibility = "hidden";
document.getElementById("mytable").style.visibility = "hidden";
ptInit(withTimetable, 'ger', 'ptf');
```

Hier wird zunäächst der Parameter "selectedIndex" für die Auswahlbox auf den Wert "-1" gesetzt. Dies bewirkt dass kein Element vorausgewählt ist.
Anschließend werden das <canvas>-Element und das <table>-Element unsichtbar geschaltet. In der Funktion "show" haben wir ja dafür gesorgt dass das jeweils benötigte HTML-Element angezeigt wird.
In der Funktion "ptInit" haben wir den letzten Parameter entfernt. Da wir keine Vorauswahl mehr treffen muss die Funktion "show" nach der Initialiserung nicht mehr aufgerufen werden.

Anzeigebeispiel - so sieht's aus: www.ptraffic.net/ptraffic-library/Beispiele/Schnelleinstieg4.htm

3. Anleitung

Hier finden Sie eine genaue Anleitung zur Nutzung der PTraffic-Bibliothek.

3.1. Verzeichnis-Struktur

Für Projekte die mit der PTraffic-Bibliothek arbeiten ist es empfehlenswert ein neues Verzeichnis für die HTML-Dateien zu erstellen. Darin können dann die 3 Unterverzeichnisse "images", "javascript" und "ptf" erstellt werden. Die Verzeichnisstruktur sieht dann beispielsweise so aus:

- MeineWebsite
  - images
  - javascript
  - ptf

Das Untervezeichnis "images" ist für alle Bilder der Website gedacht. Für die PTraffic-Bibliothek hat dieses Verzeichnis im Moment keine Relevanz, da in der Version 0.8 noch keine Funktionen für die Liniennetzpläne vorhanden sind.
Das Verzeichnis "javascript" soll alle Javascript-Dateien des Projekts enthalten.
Das Verzeichnis "ptf" soll die Tabellen des PTraffic bzw. LineMap-Draw-Projekts enthalten.

Natürlich können Sie nach Bedarf noch weitere Unterverzeichnisse erstellen, z. B. "css" für Stylesheet-Dateien oder "pdf" für PDF-Dateien etc.

In dieser Anleitung wird nachfolgend von dieser Verzeichnisstruktur ausgegängen. Wenn Sie eine andere Verzeichnisstruktur verwenden müssen Sie die Beispiele auf dieser Seite eventuell anpassen.

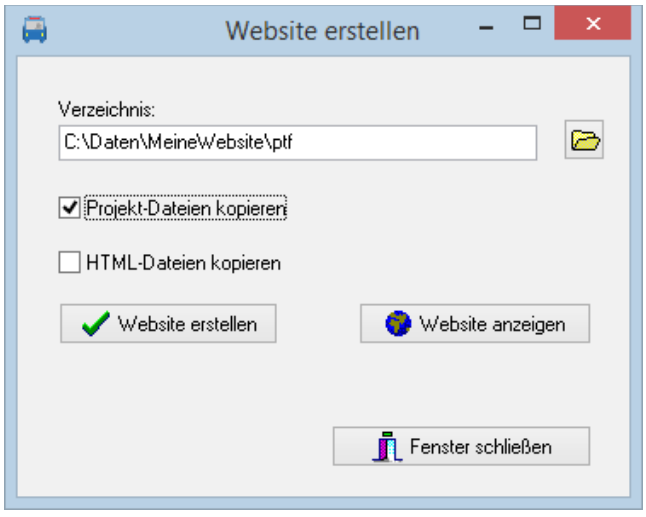
3.2. Dateien kopieren

Um eine Internet-Anwendung mit der PTraffic-Bibliothek zu erstellen müssen die Javascript-Dateien der PTraffic-Bibliothek und die Projekt-Dateien des PTraffic-Projekts kopiert werden.

3.2.1. Dateien der Javascript-Bibliothek kopieren

Laden Sie zunächst die Javascript-Bibliothek von der Startseite der PTraffic-Bibliothek unter www.ptraffic.net/ptraffic-library/index.htm herunter. Diese können dort als gepackte ZIP-Datei heruntergeladen werden. Entpacken Sie diese Dateien und kopieren Sie die Dateien in das Unterverzeichnis "javascript".

3.2.2. Projektdateien kopieren



Kopieren Sie nun die Projektdateien Ihres PTraffic-Projekts, also alle Dateien mit der Datei-Endung ".ptf" und die Projektdatei mit der Eindung ".ppr", in das Untervezeichnis "ptf".

Sie können zum Kopieren auch das PTraffic-Programm verwenden. Laden Sie das Projekt und wählen Sie den Menüpunkt "Projekt → Website erstellen". Im Fenster "Website erstellen" wählen Sie nun das Unterverzeichnis "ptf" aus und deaktivieren "HTML Dateien kopieren" (siehe Bild). Klicken Sie dann auf "Website erstellen". Allerdings wird dabei automatisch im Verzeichnis "ptf" ein

Unterverzeichnis "images" erstellt. Dort werden - soweit vorhanden - die Bilder der Liniennetzpläne kopiert. Wenn Sie die Liniennetzpläne für Ihre Anwendung benötigen können Sie diese in das eigene Verzeichnis "images" (siehe oben) kopieren und das Unterverzeichnis "images" im Verzeichnis "ptf" anschließend löschen.
In der aktuellen Version 0.8 der PTraffic-Bibliothek sind noch keine Funktionen für Liniennetzpläne vorhanden.

3.3. PTraffic-Bibliothek einbinden

Binden Sie die Javascript-Dateien der PTraffic-Bibliothek in den Dateikopf zwischen <head> und </head> ein. Wenn Ihr Projekt mit LineMap Draw erstellt wurde wird die Datei "ptraffic_timetable.js" nicht benötigt und muss daher auch nicht eingebunden werden.

Da PTraffic in der aktuellen Version kein Unicode unterstützt muss für die Zeichencodierung der HTML-Dateien das Format “iso-8859-1” eingestellt werden. Die HTML-Datei darf daher auch nicht als Unicode-Datei gespeichert werden sondern sollte als ASCII- bzw. ANSI-Datei gespeichert werden, sonst werden Umlaute und Sonderzeichen nicht korrekt dargestellt.

Die HTML-Datei würde in HTML5 beispielsweise so aussehen:

```
<!DOCTYPE html>
<html lang="de">
  <head>
    <meta charset="iso-8859-1" />
    <script src="javascript/ptraffic_text.js" type="text/javascript"></script>
    <script src="javascript/ptraffic.js" type="text/javascript"></script>
    <script src="javascript/ptraffic_timetable.js"
type="text/javascript"></script>
    <script src="javascript/publicsql.js" type="text/javascript"></script>
    <title>Meine Fahrplan-Website</title>
  </head>
<body>
...
</body>
</html
```

3.4. Initialisierung

Bevor auf die Funktionen und Variablen der PTraffic-Bibliothek zugegriffen werden kann muss diese initialisiert werden. Dazu wird die Funktion ptInit aufgerufen. Die Funktion ptInit hat folgenden Aufbau:

```
function ptInit(ptype, lang [, dir] [, fn])
```

Die Parameter haben folgende Bedeutung:

ptype: Programm mit dem die Dateien ersrtellt wurden. Mögliche Werte sind:

- withTimetable (für PTraffic-Standard; ptraffic_timetable.js muss eingebunden werden)
- withLinemap (für LineMapDraw)
- withTimetableAndLinemap (für PTraffic Pro/ProPlus; ptraffic_timetable.js muss eingebunden werden)
- withoutTimetableAndLinemap (wenn weder auf Fahrplan- noch auf Liniennetzplan-Daten zugegriffen werden muss)

lang: Sprache. gültige Werte sind (zunächst) "ger" und "eng" für deutsch und englisch.

dir: (optional) Verzeichnis in dem die PTraffic- bzw. LineMap-Draw-Dateien liegen. Wird der Parameter nicht angegeben müssen die Dateien im gleichen Verzeichnis wie die HTML-Dateien liegen

fn: (optional) Callback-Funktion. Diese wird am Ende von ptInit aufgerufen und übernimmt die weitere Verarbeitung

Der Aufruf für das Programm PTraffic Pro mit deutscher Spracheinstellung und den Projekt-Dateien im Verzeichnis "ptf" würde so aussehen:

```
ptInit(withTimetableAndLinemap, "ger", "ptf");
```

Soll direkt nach dem Aufruf auf die Bibliothek zugegriffen werden muss hierfür eine Callback-Funktion angegeben werden. Näheres dazu finden im Kapitel „Callback-Funktionen“. Der relevante Quelltext könnte dann beispielsweise so aussehen (siehe auch 2. Schnelleinstieg):

```
<div id="mydiv">
</div>

<table id="mytable" border="2">
</table>

<script>

function show() {
  showTimetableForm("mydiv", "mytable");
}

ptInit(withTimetableAndLinemap, 'ger', 'ptf', show);

</script>
```

Die Funktionsbeschreibung zu ptInit finden Sie im Kapitel „8. Funktionen“.

3.4.1. Weiterführende Hinweise

Mit diesen Informationen können Sie beginnen eigene Internet-Anwendungen mit der PTraffic-Bibliothek zu schreiben.

Zur Einarbeitung sollten Sie - falls noch nicht geschehen - die Beispiele unter „2. Schnelleinstieg“ durcharbeiten.

Im Kapitel„8. Funktionen“ finden Sie Erläuterungen zu allen Funktionen der Bibliothek. Am einfachsten zu verwenden sind Funktionen deren Name mit "show" beginnt. Diese können direkt in die Website eingebunden werden.

Für das Verständnis der anderen Funktionen sowie der Variablen (siehe Kapitel „7. Variablen“) sind Kenntnisse des Aufbaus der Tabellen nötig. Das Datenmodell für PTraffic und PTraffic Pro/ProPlus finden Sie als PDF-Datei unter www.ptraffic.net/ptraffic_download.htm (Überschrift "Dokumentation").
Für LineMap Draw gilt ebenfalls das Datenmodell für PTraffic Pro/ProPlus - allerdings ohne alle Tabellen die sich auf Fahrplandaten beziehen. Relevant sind die Tabellen "stations.ptf", "lines.ptf", "line_graphic.ptf" sowie alle Tabellen die mit "line_map" oder "lm" beginnen.

Zugriff mit ID

Die meisten Tabellen enthalten ein Feld mit dem Namen "ID" das einen Datensatz der Tabelle eindeutig identifiziert. Beispielsweise enthält die Tabelle "stations.ptf" die Felder "ID" und "Station". Die ersten 3 Datensätze für das Beispielprojekt "Musterdorf" haben beispielsweise folgenden Inhalt:

ID	Station
1	Bahnhofstraße

2	Campingplatz
3	Cityparkplatz

Es ist meistens besser über die ID auf den eigentlichen Inhalt - in diesem Fall auf den Stationsnamen - zuzugreifen, da die ID immer eindeutig ist. Der Stationsname kann sich jedoch auch ändern und man müsste dann den ganzen Quelltext entsprechend anpassen. Für den Zugriff auf den Namen kann dann die entsprechende Funktion, in diesem Fall getStationName, verwendet werden.

4. Callback-Funktionen

Viele Funktionen der PTraffic-Bibliothek sind sogenannte Callback-Funktionen. Dies bedeut das der PTraffic-Funktion eine Funktion als Parameter übergeben wird die für die weitere Ausführung nach dem Abarbeiten der PTraffic-Funktion zuständig ist.

Um mit diesen Callback-Funktionen zu arbeiten sind einige Dinge zu beachten die nachfolgend erläutert werden.

4.1. Technischer Hintergrund

Die verwendete Javascript-Bibliothek PublicSQL ermöglicht SQL-Abfragen direkt in Javascript. Die Tabellen – standardmäßig erkennbar an der Endung ".ptf" - müssen dazu nicht in die HTML-Datei eingebunden werden sondern werden beim Aufruf der SQL-Anweisungen nachträglich in die HTML-Datei eingebunden.

Da HTML und Javascript standardmäßig kein Nachladen von Dateien unterstützen wird hierzu ein Trick angewandt: Die PTF-Dateien sind eigentlich Javascript-Dateien in denen die Daten als Array gespeichert sind. Wird nun eine SQL-Abfrage durchgeführt werden alle benötigten und noch nicht geladenen Dateien nachträglich in den Dateikopf zwischen <head>. und </head>) eingebunden. Die neue Zeile sieht beispielsweise so aus:

```
<script src="ptf/stations.ptf?x=-156450406" type="text/javascript"></script>
```

Wie man sieht hat die PTF-Datei – die ja wie gesagt eine Javascript-Datei ist – einen Parameter (x=-156450406). Dieser Parameter wird aus dem aktuellen Datum und der Zeit errechnet. Dadurch wird verhindert, dass die Datei – soweit vorhanden - aus dem Cache (Zwischenspeicher) verwendet wird: Bei jeden Aufruf wird ein anderer Wert angehängt und die Datei wird nicht als gleiche Datei behandelt und daher neu geladen.

Anschließend wird dem neuen <script>-Tag ein onload-Event-Handler hinzugefügt, der aufgerufen wird sobald die Datei (im Beispiel "stations.ptf") vollständig geladen ist. In diesem Event-Handler wird zum Schluss die übergebene Callback-Funktion aufgerufen. Dadurch ist sichergestellt dass beim Aufruf der Callback-Funktion die Daten der PTF-Datei vollständig geladen und in PublicSQL eingebunden sind.

Würde man keine Callback-Funktion verwenden sondern einfach nach Ausführung der SQL-Anweisung versuchen das Ergebnis der Abfrage zu verwenden, würde dies zu Fehlern führen da das Ergebnis noch nicht zur Verfügung steht.

Der Vorteil der Callback-Funktionen besteht darin, dass ein Nachladen einzelner Dateien möglich ist. Möchte man beispielsweise einen Fahrplan ausgeben werden nur die Dateien für die gewählte Linie, Richtung sowie für den gewählten Wochentag geladen. Ansonsten müßte man bereits vorher sämtliche Fahrplandateien laden. Bei 10 Linien mit 3 Wochentag-Gruppen (Mo-Fr, Sa, So) und 2 Richtungen wären das beispielsweise 70 PTF-Dateien.

4.2. Auswirkungen

Beachtet man die Besonderheiten der Callback-Funktion nicht kann dies zu seltsamen Fehlern führen.
Im folgenden Beispiel werden 2 PublicSQL-Anweisungen direkt hintereinander aufgerufen.

```
<div id="testdiv">
</div>

<div id="names">
<b>Stations-Namen:<br></b>
</div>

<div id="ids">
<b>Stations-ID's:<br></b>
</div>

<script>

function cbfunc1(t) {
  document.getElementById('names').innerHTML += "<br>" + t;
}

function cbfunc2(t) {
  document.getElementById('ids').innerHTML += "<br>" + t;
}

publicSQL.tablePath = "ptf/"; // Pfad für PTF-Dateien
publicSQL.setTableCache(false); // Table-Cache ausschalten
publicSQL.query("select Station from stations order by Station", "cbfunc1");
publicSQL.query("select ID from stations order by ID", "cbfunc2");

</script>
```

Nach Ausführung der publicSQL.query-Anweisungen werden im <head>-Bereich der HTML-Seite gleich 2 <script>-Anweisungen eingefügt:

```
<head>
 …
<script src="ptf/stations.ptf?x=-156529607" type="text/javascript"></script>
<script src="ptf/stations.ptf?x=-156529607" type="text/javascript"></script>
</head>
```

Dies liegt daran, dass beim 2. Aufruf von publicSQLquery der 1. Aufruf noch nicht abgearbeitet wurde. Beim 2. Aufruf von PublicSQL.query ist die Datei “stations.ptf” noch nicht geladen und die Funktion cbfunc1 wurde ebenfalls noch nicht ausgeführt.

Der Umgang mit Callback-Funktionen erfordert also eine andere Vorgehensweise.

4.3. Umgang von Callback-Funktionen bei Nutzung der PTraffic-Library

Zur Vermeidung von Fehlern bei der Verwendung von Callback-Funktionen wird folgende Vorgehensweise empfohlen:

1. Wenn eine PTraffic- oder PublicSQL-Funktion aufgerufen wird, die eine Callback-Funktion als Parameter erfordert, sollte der Quelltext an dieser Stelle beendet werden. Alle weiteren Anweisungen werden dann in der Callback-Funktion geschrieben.

2. Sollen mehrere Funktionen mit übergebeneer Callback-Funktion nacheinander aufgerufen werden wird jeder Aufruf in einer eigenen Funktion geschrieben. Statt

```
Funktion1(..., "callback_func1")
...
```

```
// diverse Anweisungen nach Aufruf von Funktion1
...
Funktion2(..., "callback_func2")
...
// diverse Anweisungen nach Aufruf von Funktion2
...
Funktion3(..., "callback_func3")
...
// diverse Anweisungen nach Aufruf von Funktion3
...
// letzte Anweisung
```

sieht der Quelltext dann folgendermaßen aus:

```
function callback_func3 {
  ...
  // diverse Anweisungen nach Aufruf von Funktion3
  ...
  // letzte Anweisung
}

function callback_func2 {
  ...
  // diverse Anweisungen nach Aufruf von Funktion2
  ...
  Funktion3(..., "callback_func3")
}

function callback_func1 {
  ...
  // diverse Anweisungen nach Aufruf von Funktion1
  ...
  Funktion2(..., "callback_func2")
}

Funktion1(..., "callback_func1")
```

Es gibt noch eine weitere Methode zum Umgang mit aufeinanderfolgenden Callback-Funktionen die etwas komplizierter zu handhaben ist. Diese kann man sich in der PTraffic-Library beispielsweise bei der Funktion ptInit() anschauen. Der Vorteil dieser Methode besteht darin dass der gesamte Quelltext, durch Verwendung von Unterfunktionen, in einer einzigen Funktion geschrieben werden kann.

3. Für eine bessere Übersicht ist es empfehlenswert für alle zusammenhängenden Callback-Funktionen identische Namen mit fortlaufender Nummerierung zu verwenden. Im Beispiel wurde der Name “callback_func” mit fortlaufender Nummerierung verwendet. Durch die Namensgleichheit ist später sofort erkennbar welche Funktionen logisch zusammengehören.

Ob diese Funktionen in der oben verwendeten Reihenfolge (die erste Funktion “callback_func1” steht an letzter Stelle) oder in umgekehrte Reihenfolger (die erste Funktion steht an erster Stelle) stehen ist für die Funktionalität nicht relevant. Man sollte jedoch der Übersicht halber die Reihenfolge der Funktinosaufrufe innerhalb der Callback-Funktions-Blöcke immer gleich behandeln.

4. Am einfachsten ist es wenn man die HTML-Datei so aufbaut, dass Aufrufe von Funktionen mit übergebener Callback-Funktion nur im onload-event-handler (z. B.: <body onload=”initFunc()”>) sowie in anderen Eventhändlern für Buttons, Eingabefeldern etc. stattfinden.
Da man – z. B. mit der Funktion document.getElementById – auf alle Elemente der HTML-Datei zugreifen kann und diese auch ändern bzw. erweitern kann, ist ein Aufruf der Funktionen mit Callback-Funktions-Parameter beim Aufbau der HTML-Datei (also vor dem onload-Event)

normalerweise nicht nötig.
Da in der PTraffic-Bibliothek als erstes die Funktion ptInit() aufgerufen werden muss und diese auch eine Funktion mit Callback-Funktions-Parameter ist sollte diese im onload-event aufgerufen werden. Gegebenenfalls können dann weitere Funktionen wie oben beschrieben hinzugefügt werden.

5. Die Javascript-Funktion document.write(...) sollte grundsätzlich – nicht nur bei Verwendung von Callback-Funktionen – nur beim Aufbau der Seite verwendet werden. Sobald die Seite geladen ist sollte die Funktion nicht mehr verwendet werden. Neue Elemente (Texte, HTML-Befehle etc.) können ggfs. mit Javascript-Funktionen wie createTextnode() , createElement() und createAttribute() nachträglich hinzugefügt werden.

4.4. Unterschied bei Callback-Funktionen zwischen PublicSQL und der PTraffic-Bibliothek

In der PTraffic-Bibliothek wird der Name der Callback-Funktion direkt als Funktionsparameter übergeben. In PublicSQL wird der Name der Callback-Funktion als String übergeben.

Beispiele:

```
PTraffic:    ptInit("Pro", "ptf", CallBackFunktion);
PublicSQL:   publicSQL.query("select * from stations", "CallBackFunktion");
```

5. Einschränkungen und bekannte Fehler

Hier finden Sie Informationen zu bekannten Einschränkungen und Fehlern der aktuellen Version 0.8 der PTraffic-Bibliothek. Sollten nach Fertigstellung dieses Dokuments weitere relevante Einschränkungen und Fehler auftauchen werden diese unter www.ptraffic.net/ptraffic-library/Informationen_Einschraenkungen.htm veröffentlicht.

5.1. Fehler (ohne Auswirkung) in SQL-Abfrage

10.04.18, Datei "ptraffic.js", Zeile 132:

```
publicSQL.query('SELECT ID, Interval FROM intervalsORDER BY
Interval','ptInitIntervals');
```

Bei "intervalsORDER" fehlt das Leerzeichen vor "ORDER". Richtig wäre:

```
publicSQL.query('SELECT ID, Interval FROM intervals ORDER BY
Interval','ptInitIntervals');
```

5.2. PublicSQL-Version

10.04.18, Datei "publicsql.js":

Die Javascript-Bibliothek verwendet eine angepasste Version die nicht mit der Version 1.2.1 auf der Website www.publicsql.org übereinstimmt. Grund dafür ist ein Fehler in der Version 1.2.1 der dazu führt, dass es zu Fehlern bei 2 direkt aufeinanderfolgenden SQL-Abfragen kommen kann.

5.3. Formulare

16.04.18: In den Funktionen die eine Eingabemaske bzw. ein Eingabeformular erstellen - dies sind die meisten Funktionen deren Namen mit "show" beginnt - werden in der aktuellen Version identische Namen für die Formulare und Formularfelder verwendet. Dadurch können in den meisten Fallen keine 2 Formulare auf einer Seite gleichzeitig angezeigt werden.

In den meisten Fällen wird man ohnehin nur eine Funktion mit Eingabemaske gleichzeitig auf einer HTML-Seite benutzen. Gegebenenfalls müssen die Bezeichner für Formulare und Formularfelder in den entsprechenden Funktionen im Quelltext geändert werden.

Keine Probleme gibt es wenn man mehrere dieser Funktionen auf der Seite verwendet, für die Funktionen aber den gleichen Anzeigebereich (<DIV>-Bereich) angibt, da dieser automatisch vor Aufruf der Funktion gelöscht wird, so dass die Formular- und Feld-Bezeichner immer eindeutig sind.

6. Tipps und Tricks

6.1. Unicode statt ANSI-Zeichensatz

Da PTraffic in der aktuellen Version kein Unicode unterstützt muss für die Zeichencodierung der HTML-Dateien das Format “iso-8859-1” eingestellt werden und die HTML-Dateien müssen entsprechend als ASCII- bzw. ANSI-Datei gespeichert werden (siehe Informationen_Anleitung.htm unter "PTraffic-Bibliothek einbinden").

Möchte man stattdessen mit Unicode-Dateien arbeiten kann man die relevanten Dateien des PTraffic-Projekts in UTF8 (= Unicode) umwandeln. Dies ist mit den meisten Texteditoren einfach möglich. Mit dem kostenlosen Editor Notead++ kann man beispielsweise die Datei laden und anschließend im Menü "Kodierung" den Untermenüpunkt "Konvertiere zu UTF-8" auswählen. Speichert man dann die Datei wird diese in UTF-8-Format umgewandelt.
Allerdings darf man hierzu keine Origianldateien verwenden mit denen man später in PTraffic weiterarbeiten möchte (was normalerweise kein Problem ist, da man die Dateien ohnehin in die Website kopiert).

Es ist nicht nötig ALLE Dateien in UTF-8 umzuwandeln sondern es reicht völlig aus wenn die Dateien, die Texte mit Sonderzeichen enthalten (könnten), umgewandelt werden. Dies sind folgende Dateien:

- daygroups.ptf
- lines.ptf
- stations.ptf
- prefs.ptf (nur wenn man relevante Sonderzeichen im Projektnamen verwendet hat)
- line_maps.ptf (Es sei denn man hat nur Namen für Liniennetzpläne ohne relevante Sonderzeichen verwendet)
- (line_map_prefs: enthält Schriftnamen z. B. "Arial". Schriftnamen enthalten normalerweise keine relevanten Sonderzeichen so dass die Datei auch nicht umgewandelt werden muss)

Die Angabe im Dateikopf für den Zeichensatz lautet dann nach HTML5-Standard:

```
<meta charset="utf-8">
```

7. Variablen

Auf die nachfolgend aufgeführten Variablen kann nach dem Aufruf von ptInit() lesend(!) zugegriffen werden. Die Variableninhalte dürfen nicht geändert werden da diese von vielen Funktionen der PTraffic-Bibliothek benutzt werden.

In den meisten Fällen ist es nicht nötig und auch nicht sinnvoll auf die Variablen direkt zuzugreifen, da entsprechende Funktionen existieren. Die Wahrschinlichkeit dass sich Variablen in späteren Versionen ändern ist wesentlich größer als dass sich entsprechende Funktionen ändern. Außerdem besteht die Gefahr dass man aus Versehen den Inhalt einer Variable überschreibt und dadurch Fehler entstehen.

Die Inhalte der meisten Variablen für das Beispielprojekt "Musterdorf" können beim Test der Funktion ptInit unter www.ptraffic.net/ptraffic-library/Funktionen_Allgemein.htm eingesehen werden.

7.1. actLanguage

Typ: string

Die Variable actLanguage enthält das 3-stelliges Kürzel der aktuell eingestellten Sprache. Aktuell werden die Sprachen Deutsch und Englisch unterstützt. Die entsprechenden Kürzel lauten "ger" und "eng".Weitere Sprachen können durch Anpassung der Datei "ptraffic_text.js" hinzugefügt werden.

Statt direkt auf actLanguage zuzugreifen kann die Funktion getLanguage() genutzt werden.

7.2. stations[]

Typ: Array von Objekten

In stations sind alle Stationen in alphabetischer Reihenfolge enthalten. Ein Element des Arrays enthält ein Objekt mit den Eigenschaften "ID" und "Station". Mit stations[0].ID wird beispielsweise auf die ID des ersten Elements zugegriffen, Mit stations[3].Station auf den Namen der vierten Station.

Um auf Stationen zuzugreifen können die Funktionen getStationID und getStationName verwendet werden.

7.3. stationsByIdArray[]

Typ: Array

Das Array stationsByIdArray enthält für alle Station-ID's den zugehörigen Index des Arrays stations[]. Dabei ist zu beachten dass die niedrigste Station-ID nicht 0 sondern 1 ist. Außerdem kann es vorkommen dass Station-IDs nicht fortlaufend vorhanden sind. Dies ist der Fall wenn nachträglich Stationen gelöscht wurden. Daher ist es nicht ohne weiteres möglich mit einer Schleife alle Elemente des Arrays zu durchlaufen.
Beispiel: Mit stationsByIdArray[4] erhalten wir den Index der Stations-ID 4 des Arrays stations[]. Steht in stationsByIdArray[4] z. B. der Wert 7 können wir mit stations[7].Station auf den Namen der Station zugreifen.

Um über die Station-ID auf den Stationsnamen zuzugreifen kann die Funktion getStationName verwendet werden.

7.4. stationsByNameArray[]

Typ: Array

Das Array stationsByNameArray enthält für alle Stations-Namen in alphabetischer Reihenfolge den

zugehörigen Index des Arrays stations[].
Beispiel: Mit stationsByNameArray["Hauptbahnhof"] erhalten wir den Index der Station im Array stations[]. Liefert stationsByNameArray["Hauptbahnhof"] z. B. der Wert 7 können wir mit stations[7].ID auf die Station-ID zugreifen.

Um über den Stationsnamen auf die Station-ID zuzugreifen kann die Funktion getStationId verwendet werden.

7.5. lines[]

Typ: Array von Objekten

In lines sind alle Linien in alphabetischer Reihenfolge enthalten. Ein Element des Arrays enthält ein Objekt mit den Eigenschaften "ID" und "Line". Mit lines[0].ID wird beispielsweise auf die ID des ersten Elements zugegriffen, Mit lines[3].Line auf den Namen der vierten Linie.

Um auf Linien zuzugreifen können die Funktionen getLineID und getLineName verwendet werden.

7.6. linesByIdArray[]

Typ: Array

Das Array linesByIdArray enthält für alle Line-ID's den zugehörigen Index des Arrays lines[]. Dabei ist zu beachten dass die niedrigste Line-ID nicht 0 sondern 1 ist. Außerdem kann es vorkommen dass Line-IDs nicht fortlaufend vorhanden sind. Dies ist der Fall wenn nachträglich Linien gelöscht wurden. Daher ist es nicht ohne weiteres möglich mit einer Schleife alle Elemente des Arrays zu durchlaufen.
Beispiel: Mit linesByIdArray[4] erhalten wir den Index der Line-ID 4 des Arrays lines[]. Steht in linesByIdArray[4] z. B. der Wert 7 können wir mit lines[7].Line auf den Namen der Linie zugreifen.

Um über die Line-ID auf den Linien-Namen zuzugreifen kann die Funktion getLineName verwendet werden.

7.7. linesByNameArray[]

Typ: Array

Das Array linesByNameArray enthält für alle Linien-Namen in alphabetischer Reihenfolge den zugehörigen Index des Arrays lines[].
Beispiel: Mit linesByNameArray["Linie 1"] erhalten wir den Index der Station im Array lines[]. Liefert linesByNameArray["Linie 1"] z. B. der Wert 2 können wir mit lines[2].ID auf die Station-ID zugreifen.

Um über den Linien-Name auf die Linien-ID zuzugreifen kann die Funktion getLineId verwendet werden.

7.8. lineStations[]

Typ: Array von Objekten

In lineStations werden alle Linienverläufe gespeichert. Dazu enthält das Array Objekte mit den Feldern LineID, Position und StationID. In dieser Reihenfolge ist das Array auch sortiert. Das Feld Position bezeichnet die Position der Station innerhalb der Linie. Wenn im Programm bei einer Linie eine Station 2mal hintereinander angegeben wurde (für Ankunft und Abfahrt) wird dies auch in linestations übernommen.

Statt direkt auf lineStations[] zuzugreifen können folgende Funktionen verwendet werden:

Funktion	Beschreibung
getStationLineIds	Gibt Array mit Linien-ID's der übergebenen Stations-ID zurück.
getStationLineNames	Gibt Array mit Linien-Namen der übergebenen Stations-ID zurück.
getLineStationIds	Gibt Array mit Station-ID's der übergebenen Linien-ID zurück.
getLineStationNames	Gibt Array mit Stations-Namen der übergebenen Linien-ID zurück.

7.9. LineStationsByLineIdArray[][]

Typ: 2-dimensionales Array

Das Array lineStationsByLineIdArray erleichtert den Zugriff auf den Linienverlauf einer bestimmten Linie. Die 1. Dimension enthält die Linien-ID's, die 2. Dimension die Position. Der enthaltene Wert ist der Index des Arrays stations[]. lineStationsByLineIdArray[1][3] liefert beispielsweise den Index der 4. Station der Linie mit der Linien-ID 1 des Array stations[].

Statt direkt auf lineStationsByLineIdArray[] zuzugreifen können folgende Funktionen verwendet werden:

Funktion	Beschreibung
getLineStationIds	Gibt Array mit Station-ID's der übergebenen Linien-ID zurück.
getLineStationNames	Gibt Array mit Stations-Namen der übergebenen Linien-ID zurück.

7.10. stationLinesByStationIdArray[][]

Typ: 2-dimensionales Array

Das Array stationLinesByStationIdArray erleichtert den Zugriff auf alle Linien einer bestimmten Station. Die 1. Dimension enthält die Stations-ID's, die 2. Dimension enthält die laufende Nummer beginnend bei 0. Der enthaltene Wert ist der Index des Arrays lines[]. stationLinesByStationIdArray[3][1] liefert beispielsweise den Index der 2. Linie für die Station mit der Station-ID 3.im Array lines[].

Statt direkt auf stationLinesByStationIdArray[] zuzugreifen können folgende Funktionen verwendet werden:

Funktion	Beschreibung
getStationLineIds	Gibt Array mit Linien-ID's der übergebenen Stations-ID zurück.
getStationLineNames	Gibt Array mit Linien-Namen der übergebenen Stations-ID zurück.

7.11. dayGroups[]

Typ: Array von Objekten

In dayGroups sind alle Wochentag-Gruppen (Fahrtage) enthalten. Ein Element des Arrays enthält ein Objekt mit folgenden Eigenschaften:

Name	Bedeutung
ID	Eindeutiger Index für Zugriff auf die Tag-Gruppe

Group	Name der Wochentag-Gruppe (z. B. "MO-FR")
Days	7-Zeichen-String (ein Zeichen je Wochentag beginnend bei Montag). „1“ bedeutet die Linie fährt an diesem Tag der Wert „0“ bedeutet die Linie fährt nicht. Beispiel: "1111100" steht für montags-freitags.
Text	anzuzeigender Text für die Tag-Gruppe, z. B. "montags - freitags"

Die Eigenschaften sind äquivalent zur Tabelle daygroups.ptf. Die Sortierung erfolgt nach dem Feld "Days", Mit dayGroups[0].Text wird beispielsweise auf den Text der ersten Wochentag-Gruppe zugegriffen.

Um auf Wochentag-Gruppen zuzugreifen stehen zur Zeit noch keine Funktionen zur Verfügung.

7.12. dayGroupsByIdArray[]

Typ: Array

Das Array dayGroupsByIdArray enthält für alle Daygroup-ID's den zugehörigen Index des Arrays dayGroups[]. Dabei ist zu beachten dass die niedrigste Daygroup-ID nicht 0 sondern 1 ist. Außerdem kann es vorkommen dass Daygroup-IDs nicht fortlaufend vorhanden sind. Dies ist der Fall wenn nachträglich Wochentag-Gruppen gelöscht wurden. Daher ist es nicht ohne weiteres möglich mit einer Schleife alle Elemente des Arrays zu durchlaufen.
Beispiel: Mit dayGroupsByIdArray[1] erhalten wir den Index der Daygroup-ID 1 des Arrays dayGroups[]. Steht in dayGroupsByIdArray[1] z. B. der Wert 2 können wir mit dayGroups[2].Text auf den Text für die Wochentag-Gruppe zugreifen.

Um auf Wochentag-Gruppen zuzugreifen stehen zur Zeit noch keine Funktionen zur Verfügung.

7.13. lineConnections[]

Typ: Array von Objekten

In lineConnections sind alle vorhandenen Fahrpläne des Projekts eingetragen. Ein Element des Arrays enthält ein Objekt mit folgenden Eigenschaften:

Name	Bedeutung
ID	Lineconnection-ID (eindeutiger Index für Zugriff auf den Fahrplan)
LineID	Linien-ID (eindeutiger Index der Linie)
DayGroupID	Daygroup-ID (eindeutiger Index der Daygroup)
Direction	Richtung (1 oder 2)

Die Eigenschaften sind äquivalent zur Tabelle line_connections.ptf. Die Sortierung erfolgt nach den Feldern "LineID", "DayGroupID" und "Direction" (in der Reihenfolge). Mit lineConnections[2].Direction wird beispielsweise auf die Richtung des 3. Eintrags des Array lineConnectinos zugegriffen.

Mit der Funktion getLineConnectionId erhält man die LineConnection-ID für die übergebene Linie, Tag-Gruppe und Richtung.

7.14. lineConnectionsByIdArray[]

Typ: Array

Das Array lineConnectionsByIdArray ermöglicht den Zugriff mit der LineConnection-ID auf die Felder "LineID", DayGroupID" und "Directions" des Arrays lineConnections[]. Der enthaltene Wert ist der Index des Array lineConnections[]. lineConnectionsByIdArray[3] liefert beispielsweise den Index des Arrays lineConnections mit der LineConnection-ID 3.

Mit der Funktion getLineConnectionId erhält man die LineConnection-ID für die übergebene Linie, Tag-Gruppe und Richtung.

7.15. lineConnectionsByLineIdArray[][]

Typ: 2-dimensionales Array

Das Array lineConnectionsByLineIdArray ermöglicht den Zugriff auf alle vorhandenen Fahrpläne einer bestimmten Linie. Die 1. Dimension enthält die Linien-ID's, die 2. Dimension enthält die laufende Nummer beginnend bei 0. Der enthaltene Wert ist der Index des Array lineConnections[]. lineConnectionsByLineIdArray[3][1] liefert beispielsweise den Index des Arrays lineConnections für den 2. vorhandenen Fahrplan für die Linie mit der Linien-ID 3.

Mit der Funktion getLineConnectionId erhält man die LineConnection-ID für die übergebene Linie, Tag-Gruppe und Richtung.

7.16. intervals[]

Typ: Array von Objekten

Das Array intervals enthält die Interval-Zeiten für die vorhandenen Fahrpläne. Ein Element des Arrays enthält ein Objekt mit folgenden Eigenschaften:

Name	Bedeutung
ID	Eindeutiger Index für Zugriff auf den Interval-Eintrag.
Interval	Interval-Zeit (z. B. „20“ für „fährt alle 20 Minuten“).

Die Eigenschaften sind äquivalent zur Tabelle intervals.ptf. Die Sortierung erfolgt aufsteigend nach dem Feld "Interval", Mit intervals[0].Intervals wird beispielsweise auf die Intervall-Zeit des ersten Eintrags zugegriffen. Die kleinste mögliche Intervall-ID ist 2 und nicht wie sonst üblich 1.

Um auf die Intervall-Zeit einer bestimmten Intervall-ID zuzugreifen kann die Funktion getInterval genutzt werden.

7.17. intervalsByIdArray[]

Typ: Array

Das Array intervalsByIdArray enthält für alle Intervall-ID's den zugehörigen Index des Arrays intervals[]. Dabei ist zu beachten dass die niedrigste Intervall-ID nicht 0 sondern 2 ist. Außerdem kann es vorkommen dass Intervall-IDs nicht fortlaufend vorhanden sind. Dies ist der Fall wenn nachträglich Intervall-Zeiten gelöscht wurden. Daher ist es nicht ohne weiteres möglich mit einer Schleife alle Elemente des Arrays zu durchlaufen.
Beispiel: Mit intervalsByIdArray[4] erhalten wir den Index der Intervall-ID 4 des Arrays intervals[]. Steht in intervalsByIdArray[4] z. B. der Wert 2 können wir mit intervals[2].Interval auf die Intervall-Zeit zugreifen.

Um auf die Intervall-Zeit einer bestimmten Intervall-ID zuzugreifen kann die Funktion getInterval genutzt werden.

8. Funktionen

Hier finden Sie Informationen zu alle Funktionen der PTraffic-Bibliothek. Unter www.ptraffic.net/ptraffic-library/Funktionen.htm können Sie die Funktionen auch direkt testen.

8.1. Allgemeine Funktionen

8.1.1. ptInit

Initialisiert die PTraffic Bibliothek. Die Funktion muss aufgerufen werden bevor auf Funktionen und Variablen der PTraffic-Bibliothek zugegriffen wird.

Aufbau

function ptInit(ptype, lang [, dir] [, fn])

Parameter

ptype: Programm mit dem die Dateien ersrtellt wurden. Mögliche Werte sind:

- withTimetable (für PTraffic-Standard; ptraffic_timetable.js muss eingebunden werden)
- withLinemap (für LineMapDraw)
- withTimetableAndLinemap (für PTraffic Pro/ProPlus; ptraffic_timetable.js muss eingebunden werden)
- withoutTimetableAndLinemap (wenn weder auf Fahrplan- noch auf Liniennetzplan-Daten zugegriffen werden muss)

lang: Sprache. gültige Werte sind (zunächst) "ger" und "eng" für deutsch und englisch. Es können weitere Sprachen hinzugefügt werden. Verwendet werden sollte jeweils der 3-stellige ISO 639-2 Code.

dir: (optional) Verzeichnis in dem die PTraffic- bzw. LineMap-Draw-Dateien liegen. Wird der Parameter nicht angegeben müssen die Dateien im gleichen Verzeichnis wie die HTML-Dateien liegen

fn: (optional) Callback-Funktion. Diese wird am Ende von ptInit aufgerufen und übernimmt die weitere Verarbeitung.

8.1.2. setLanguage

Legt die aktuelle Sprache fest.

Aufbau

function setLanguage(lang)

Parameter

lang: 3-stelliges Kürzel der Sprache. Aktuell sind die Werte "ger" (deutsch) und "eng" (englisch) möglich; Die Sprache muss in ptraffic_text.js definiert sein. Als Kürzel sollte der zugehörigeISO 639-2 Code der Sprache verwendet werden.
Hinweis: Für die HTML-Seiten sollte"iso-8859-1" eingestellt werden, da PTraffic aktuell noch kein Unicode unterstützt.

8.1.3. getLanguage

Gibt die aktuelle Sprache zurück.

Aufbau

function getLanguage()

Parameter

Rückgabe: 3-stelliges Kürzel der Sprache; Sprachen werden in ptraffic_text.js definiert. Als Kürzel wird der ISO 639-2 Code der Sprache verwendet werden.
Hinweis: Für die HTML-Seiten sollte"iso-8859-1" eingestellt werden, da PTraffic aktuell noch kein Unicode unterstützt.

8.1.4. showLanguageSelectDialog

Zeigt Sprachauswahl-Dialog zum Wechseln der verwendeten Sprache.

Aufbau

function showLanguageSelectDialog(did, update [, selfLang] [, fn])

Parameter

did: enthält die <DIV>-ID zur Anzeige des Sprachauswahl-Dialogs.

update:
true = Der Sprachauswahldialog wird beim Wechseln der Sprache in der neu gewählten Sprache angezeigt
false = Der Sprachauswahldialog wird beim Wechseln der Sprache weiterhin in der ursprünglich gewählten Sprache angezeigt

selfLang:
true = Die Sprachen in der Auswahlbox werden in den jeweiligen Landessprachen angezeigt - unabhängig von der aktuell gewählten Sprache. Bei den beiden enthaltenen Sprachen enthält die Auswahlbox demzufolge immer die Werte deutsch" und "english".
false = Die Sprachen werden in gleichen Sprache angezeigt wie der Sprachauswahldialog (Voreinstellung wenn Parameter nicht vorhanden ist)

fn: Funktion die aufgerufen wird wenn die Sprache geändert wird.

8.2. Funktionen für Stationen

8.2.1. GetStationName

Gibt Stationsnamen der übergebenen Stations-ID zurück.

Aufbau

function getStationName(sid)

Parameter

sid: Stations-ID für die der Name gesucht wird

Rückgabe: Name der Station

8.2.2. getStationId

Gibt Stations-ID des übergebenen Stationsnamens zurück.

Aufbau

function getStationId(sn)

Parameter

sn: Stations-Name für den die Stations-ID gesucht wird

Rückgabe: Stations-ID

8.2.3. showStations

Aufbau

function ShowStations([tb])

Parameter

tb: (optional) enthält die Tabelle zur Anzeige der Stationen. Mögliche Werte:

- die ID des <table>-Elements als String
- den Index der Tabelle (0 für die erste Tabelle, 1 für die 2. Tabelle, usw.)
- das Tabellenelement als Object.

Wird kein Wert übergeben erstellt die Funktion eine neue Tabelle mit der in publicSQL.htmlStandardTable enthaltenen <table>-ID.
Existiert bereits eine Tabelle mit dieser ID, wird diese geleert und für die Anzeige genutzt.

8.3. Funktionen für Linien

8.3.1. getLineName

Gibt den Linien-Namen der übergebenen Linien-ID zurück.

Aufbau

function getLineName(lid)

Parameter

lid: die Linien-ID für die der Name gesucht wird

Rückgabe: Linien-Name der übergebenen Linien-ID

8.3.2. getLineId

Gibt die Linien-ID für den übergebenen Linien-Namen zurück.

Aufbau

function getLineId(ln)

Parameter

ln: der Linien-Name für den die Linien-ID gesucht wird

Rückgabe: Linien-ID des übergebenen Linien-Namens

8.3.3. showLines

Aufbau

function showLines([tb])

Parameter

tb: (optional) enthält die Tabelle zur Anzeige der Linien. Mögliche Werte:

- die ID des <table>-Elements als String
- den Index der Tabelle (0 für die erste Tabelle, 1 für die 2. Tabelle, usw.)

- das Tabellenelement als Object.

Wird kein Wert übergeben erstellt die Funktion eine neue Tabelle mit der in publicSQL.htmlStandardTable enthaltenen <table>-ID.
Existiert bereits eine Tabelle mit dieser ID, wird diese geleert und für die Anzeige genutzt.

8.3.4. getStationLineIds

Gibt Array mit Linien-ID's der übergebenen Stations-ID zurück.

Aufbau

function getStationLineIds(sid [, alphasort])

Parameter

sid: die Station-ID für die die Linien-ID's gesucht werden.

alphasort: (optional) Sortierung der Linien-ID's.
true = alphabetisch sortieren nach den zugehörigen Linien-Namen (Voreinstellung wenn Parameter nicht angegeben wird).
false = nach Linien-ID's aufsteigend sortieren.

Rückgabe: Array mit Linien-ID's der übergebenen Stations-ID. Bei fehlerhafter Stations-ID wird ein leeres Array zurückgegeben.

8.3.5. getStationLineNames

Gibt Array mit Linien-Namen der übergebenen Stations-ID zurück.

Aufbau

function getStationLineNames(sid [, alphasort])

Parameter

sid: die Station-ID für die die Linien-Namen gesucht werden.

alphasort: (optional) Sortierung der Linien-Namen.
true = alphabetisch sortieren (Voreinstellung wenn Parameter nicht angegeben wird).
false = nach zugehörigen Linien-ID's aufsteigend sortieren.

Rückgabe: Array mit Linien-Namen der übergebenen Stations-ID. Bei fehlerhafter Stations-ID wird leeres Array zurückgegeben.

8.3.6. getLineStationIds

Gibt Array mit Stations-ID's der übergebenen Linien-ID zurück.

Aufbau

function getLineStationIds(lid)

Parameter

lid: die Linien-ID für die Stationen gesucht werden

Rückgabe: Array mit Stations-ID's der übergebenen Linien-ID. Bei fehlerhafter Linien-ID wird ein leeres Array zurückgegeben.

8.3.7. getLineStationNames

Gibt Array mit Stations-Namen der übergebenen Linien-ID zurück.

Aufbau

function getLineStationNames(lid)

Parameter

lid: Linien-ID für die Stations-Namen gesucht werden

Rückgabe: Array mit Stations-Namen der übergebenen Linien-ID. Bei fehlerhafter Linien-ID wird leeres Array zurückgegeben.

8.3.8. showLineCourse

Zeigt eine Tabelle mit dem Lininverlauf einer Linie

Aufbau

function showLineCourse(lid [, tb])

Parameter

lid: Linien-ID der Linie für die der Linienverlauf angezeigt werden soll

tb: (optional) enthält die Tabelle zur Anzeige der Linien. Mögliche Werte:

- die ID des <table>-Elements als String
- den Index der Tabelle (0 für die erste Tabelle, 1 für die 2. Tabelle, usw.)
- das Tabellenelement als Object.

Wird kein Wert übergeben erstellt die Funktion eine neue Tabelle mit der in publicSQL.htmlStandardTable enthaltenen <table>-ID.
Existiert bereits eine Tabelle mit dieser ID, wird diese geleert und für die Anzeige genutzt.

8.3.9. showLineCourseForm

zeigt Formular zur Anzeige des Linienverlauf einer auszuwählenden Linie an

Aufbau

function showLineCourseForm(did, tid)

Parameter

did: enthält die <DIV>-ID für den Bereich zur Anzeige des Dialogformulars

tid: enthält die Tabellen-ID der Tabelle zur Anzeige des Linienverlaufs

8.4. Funktionen für Fahrpläne und Verbindungen

Hier sind Funktionen zu finden die sich auf Fahrplan-Tabellen beziehen.

Neben den nachfolgend beschriebenen Funktionen gibt es noch folgende Hilfsfunktinoen:

function minutesToTimeString(m)

Wandelt Minuten-Wert in Zeitstring um

m: Zeit in Minuten (für 01:30 Uhr wird beispielsweise 90 übergeben)

Rückgabe: Zeitstring im Format "HH:MM"

function getMinutesFromTime(t)

Gibt Minutenwert des übergebnenen Zeit-Strings zurück

t: Zeitstring im Format "HH:MM".

Rückgabe: Zeit in Minuten (für "01:30" Uhr wird beispielsweise 90 zurückgegeben)

8.4.1. getInterval

Gibt Interval der übergebenen Intervall-ID zurück

Aufbau

function getInterval(iad)

Parameter

iad: Intervall-ID für die der Intervall-Wert in Minuten gesucht wird

Rückgabe: Intervall in Minuten; -1 wenn ID nicht vorhanden

8.4.2. getLineConnectionId

Ermittelt LineConnection-ID für übergebene Linie, Taggruppe und Richtung.

Aufbau

function getLineConnectionId(lid, dgid, direction)

Parameter

lid: die Linien-ID für die gesuchte Verbindung

dgid: Daygroup-ID für die gesuchte Verbindung

direction:: Richtung (1 oder 2) für die Verbindung

Rückgabe: LineConnection-ID für die gesuchte Verbindung. Wenn keine Verbindung vorhanden ist wird -1 zurückgegeben.

8.4.3. showTimetable

Zeigt Fahrplan für eine Linie, Tag-Gruppe und Richtung

Aufbau

function showTimetable(lid, dgid, direction, tid [, fn])

Parameter

lid: die Linien-ID für den anzuzeigenden Fahrplan

dgid: Daygroup-ID für den anzuzeigenden Fahrplan

direction:: Richtung (1 oder 2) für den anzuzeigenden Fahrplan

tid: enthält die Tabellen-ID für Tabelle in der der Fahrplan angezeigt werden soll.

fn: (optional) Callback-Funktion. Diese wird am Ende der Funktion aufgerufen und übernimmt die weitere Verarbeitung.
Wird benötigt, wenn das Ergebnis der Funktion weiterverarbeitet werden soll.

8.4.4. showTimetableForm

zeigt Fahrplan-Dialog um einen Fahrplan auszuwählen und anzuzeigen.

Aufbau

function showTimetableForm(did, tid [, fn])

Parameter

did: enthält die <DIV>-ID zur Anzeige des Fahrplandialogs

tid: enthält die Tabellen-ID zur Anzeige des Fahrplans

fn: (optional) Callback-Funktion. Diese wird am Ende der Funktion aufgerufen und übernimmt die weitere Verarbeitung.
Wird benötigt, wenn das Ergebnis der Funktion weiterverarbeitet werden soll.

8.4.5. showCurrentRides

Zeigt alle Fahrten die zur aktuellen Uhrzeit für eine bestimmte Linie, Tag-Gruppe und Richtung stattfinden als Fahrplantabelle an

Aufbau

function showCurrentRides(lid, dgid, direction, tid [, fn])

Parameter

lid: die Linien-ID für die gesuchten Fahrten

dgid: Daygroup-ID für die gesuchten Fahrten

direction:: Richtung (1 oder 2) für die gesuchten Fahrten

tid: enthält die Tabellen-ID der Tabelle in der die Fahrten angezeigt werden sollen

fn: (optional) Callback-Funktion. Diese wird am Ende der Funktion aufgerufen und übernimmt die weitere Verarbeitung.
Wird benötigt, wenn das Ergebnis der Funktion weiterverarbeitet werden soll.

8.4.6. showCurrentRidesForm

zeigt Eingabemaske um alle zum aktuellen Zeitpunkt stattfindende Fahrten anzuzeigen.

Aufbau

function showCurrentRidesForm(did, tid [, fn])

Parameter

did: enthält die <DIV>-ID zur Anzeige des Fahrplandialogs

tid: enthält die Tabellen-ID zur Anzeige der aktuellen Fahrten

fn: (optional) Callback-Funktion. Diese wird am Ende der Funktion aufgerufen und übernimmt die weitere Verarbeitung.
Wird benötigt, wenn das Ergebnis der Funktion weiterverarbeitet werden soll.

8.4.7. showTimetablePeriod

Zeigt Fahrten einer Linie Tag-Gruppe und Richtung für einen bestimmten Zeitraum an.

Aufbau

function showTimetablePeriod(lid, dgid, direction, stationPos, fromTime, toTime, tid [, fn])

Parameter

lid: die Linien-ID für die anzuzeigenden Fahrten

dgid: Daygroup-ID für die anzuzeigenden Fahrten

direction:: Richtung (1 oder 2) für die anzuzeigenden Fahrten

stationPos: Stations-Position im Fahrplan wenn die Fahrten nur für eine bestimmte Station angezeigt werden sollen.
Fahrten die an dieser Station nicht halten werden nicht berücksichtigt. 0 = alle Fahrten anzeigen.

fromTime: Zeit in Minuten.als Zahl (Number), als String, als Zeit-String im Format HH:MM oder als Date-Objekt. Ab dieser Zeit werden die Fahrten angezeigt.
Bezieht sich auf in stationPos übergebene Station wenn diese angegeben wurde.

toTime: Zeit in Minuten.als Zahl (Number), als String, als Zeit-String im Format HH:MM oder als Date-Objekt. Bis zu dieser Zeit werden die Fahrten angezeigt.
Bezieht sich auf in stationPos übergebene Station wenn diese angegeben wurde.

tid: enthält die Tabellen-ID der Tabelle in der die Fahrten angezeigt werden sollen.

fn: (optional) Callback-Funktion. Diese wird am Ende der Funktion aufgerufen und übernimmt die weitere Verarbeitung.
Wird benötigt, wenn das Ergebnis der Funktion weiterverarbeitet werden soll.

8.4.8. showTimetablePeriodForm

zeigt Eingabemaske um Fahrten einer Linie Tag-Gruppe und Richtung für einen bestimmten Zeitraum anzuzeigen.

Aufbau

function showTimetablePeriodForm(did, tid [, fn])

Parameter

did: enthält die <DIV>-ID zur Anzeige der Eingabemaske

tid: enthält die Tabellen-ID der Tabelle in der der Fahrplan angezeigt werden soll

fn: (optional) Callb ack-Funktion. Diese wird am Ende der Funktion aufgerufen und übernimmt die weitere Verarbeitung.
Wird benötigt, wenn das Ergebnis der Funktion weiterverarbeitet werden soll.

8.4.9. showGraphicalTimetable

Zeigt Bildfahrplan einer Linie, Tag-Gruppe und Richtung für einen bestimmten Zeitraum an.

Aufbau

function showGraphicalTimetable(lid, dgid, direction, fromTime, toTime, width, minHeight [, colors][, cid][, fn])

Parameter

lid: Linien-ID für den Bildfahrplan

dgid: Daygroup-ID für den Bildfahrplan

direction:: Erste Richtung (1 oder 2) für die Verbindung. Die Fahrten der ersten Richtung werden von links oben nach rechts unten gezeichnet. Falls auch Fahrten für die Gegenrichtung existieren werden diese dann von links unten nach rechts oben gezeichnet.

fromTime: Zeit in Minuten.als Zahl (Number), als String, als Zeit-String im Format HH:MM oder als Date-Objekt. Ab dieser Zeit wird der Bildfahrplan angezeigt.

toTime: Zeit in Minuten.als Zahl (Number), als String, als Zeit-String im Format HH:MM oder als Date-Objekt. Bis zu dieser Zeit wird der Bildfahrplan angezeigt.

width: Gesamtbreite für den Bildfahrplan. Die Breite des Canvas-Elements wird entsprechend angepasst.

minHeight: Mindesthöhe für den Bildfahrplan. Die Breite des Canvas-Elements wird entsprechend angepasst. Da der Platz ausreichen muss um die Stationen mit den errechneten Abständen anzuzeigen, kann die tatsächliche Höhe des Bildfahrplans größer ausfallen.

colors: (optional) Assoziatives Array mit allen nötigen Farben. Das Array (bzw. Object) muss folgendermassen aufgebaut sein:

```
var meineFarben = new Object();
meineFarben["bgColor"] = "white";             // Hintergrundfarbe
meineFarben["lineColor1"] = "green";          // Richtung 1
meineFarben["lineColor2"] = "blue";           // Richtung 2
meineFarben["lineColorStart1"] = "red";       // Richtung 1 Start
meineFarben["lineColorStart2"] = "red";       // Richtung 2 Start
meineFarben["lineColorHalt1"] = "green";      // Richtung 1 Halt
meineFarben["lineColorHalt2"] = "blue";       // Richtung 2 Halt
meineFarben["lineColorEnd1"] = "black";       // Richtung 1 Ende
meineFarben["lineColorEnd2"] = "black";       // Richtung 2 Ende
meineFarben["stationLine"] = "gray";          // Stations-Linie
meineFarben["timeLine"] = "gray";             // Zeit-Linie
meineFarben["timeStepLine"] = "gray";         // Zeit-Zwischenschritt
meineFarben["timeText"] = "black";            // Zeit-Text
meineFarben["activeStationText"] = "black";   // Stations-Name
meineFarben["inactiveStationText"] = "red";   // Stations-Name (ohne Halt)
```

Als Farben können alle gültigen HTML-Farbnamen oder Hexadezimalwerte (z. B. "#808080") verwendet werden.

Wird der Parameter nicht übergeben werden Standardfarben verwendet die im Array grapicalScheduleColors mit gleichen Aufbau enthalten sind. Dadurch ergeben sich 3 Möglichkeiten zur Beeinflussung der Farben:

1. Die Standardfarben im Array grapicalScheduleColors können direkt im Quelltext angepasst werden. Diese sind in der Datei ptraffic_timetable.js definiert. Dies hat allerdings den Nachteil dass man diese Änderungen bei einem Update der Javascript-Bibliothek erneut anpassen muss.
2. Vor Aufruf der Funktion showGraphicalTimetable können einzelne Farben des Arrays grapicalScheduleColors überschrieben werden. Beispiel:
   grapicalScheduleColors["bgColor"] = "#808080"; // grauer Hintergrund
   Diese Methode bietet sich an wenn man nur wenige Farben ändern möchte.
3. Es wird eigenes Object mit dem entsprechenden Aufbau (s.o.) erstellt und als Parameter übergeben.

cid: (optional) Enthält die Canvas-ID für den Bildfahrplan. Wird keine Canvas-ID übergeben erstellt die Funktion einen Canvas-Bereich.

fn: (optional) Callback-Funktion. Diese wird am Ende der Funktion aufgerufen und übernimmt die weitere Verarbeitung.
Wird benötigt, wenn das Ergebnis der Funktion weiterverarbeitet werden soll.

8.4.10. showGraphicalTimetableForm

Zeigt Eingabemaske um den Bildfahrplan einer Linie, Tag-Gruppe und Richtung für einen bestimmten Zeitraum anzuzeigen.

Aufbau

function showGraphicalTimetableForm(did, cid [, fn])

Parameter

did: enthält die <DIV>-ID zur Anzeige des Eingabe-Formulars für den Bildfahrplan

cid: enthält die Canvas-ID für den <CANVAS>-Bereich in dem der Bildfahrplan angezeigt werden soll.

fn: (optional) Callback-Funktion. Diese wird am Ende der Funktion aufgerufen und übernimmt die weitere Verarbeitung.
Wird benötigt, wenn das Ergebnis der Funktion weiterverarbeitet werden soll.

8.4.11. showStationTimetable

Zeigt Fahrten einer Station für ausgewählte Linie, Richtung und Fahrtage an.

Aufbau

function showStationTimetable(sid, lid, dgid, direction, tid [, fn])

Parameter

sid: die Stations-ID für den Fahrplan

lid: die Linien-ID für den Fahrplan

dgid: Daygroup-ID für den Fahrplan

direction:: Richtung (1 oder 2) für den Fahrplan

tid: enthält die Tabellen-ID für die Anzeige des Fahrplans

fn: (optional) Callback-Funktion. Diese wird am Ende der Funktion aufgerufen und übernimmt die weitere Verarbeitung.
Wird benötigt, wenn das Ergebnis der Funktion weiterverarbeitet werden soll.

8.4.12. showStationTimetableForm

Zeigt Fahrplan-Dialog um Fahrten einer Station für ausgewählte Linie, Richtung und Fahrtage anzuzeigen.

Aufbau

function showStationTimetableForm(did, tid [, fn])

Parameter

did: enthält die <DIV>-ID zur Anzeige des Dialog-Formulars.

tid: enthält die Tabellen-ID zur Anzeige der Fahrpläne.

fn: (optional) Callback-Funktion. Diese wird am Ende der Funktion aufgerufen und übernimmt die weitere Verarbeitung.
Wird benötigt, wenn das Ergebnis der Funktion weiterverarbeitet werden soll.

8.4.13. showDepartures

Zeigt die aktuellen Fahrten der zuvor ausgewählten Station für einen einstellbaren Zeitraum an.

Aufbau

function showDepartures(sid, dgid, fromTime, toTime, tid [, fn])

Parameter

sid: Stations-ID für die Abfahrts-Tabelle

dgid: Daygroup-ID für die Abfahrts-Tabelle

fromTime: Zeit in Minuten.als Zahl (Number), als String, als Zeit-String im Format HH:MM oder als Date-Objekt.
Ab dieser Zeit wird die Abfahrts-Tabelle angezeigt.

toTime: Zeit in Minuten.als Zahl (Number), als String, als Zeit-String im Format HH:MM oder als Date-Objekt.
Bis zu dieser Zeit wird die Abfahrts-Tabelle angezeigt.

tid: enthält die Tabellen-ID für die Anzeige die Abfahrts-Tabelle

fn: (optional) Callback-Funktion. Diese wird am Ende der Funktion aufgerufen und übernimmt die weitere Verarbeitung.
Wird benötigt, wenn das Ergebnis der Funktion weiterverarbeitet werden soll.

8.4.14. showDeparturesForm

zeigt Eingabemaske um die aktuellen Fahrten einer bestimmten Station anzuzeigen.

Aufbau

function showDeparturesForm(did, tid [, fn])

Parameter

did: enthält die <DIV>-ID zur Anzeige der Eingabemaske

tid: enthält die Tabellen-ID zur Anzeige der Abfahrtstabelle

fn: (optional) Callback-Funktion. Diese wird am Ende der Funktion aufgerufen und übernimmt die weitere Verarbeitung.
Wird benötigt, wenn das Ergebnis der Funktion weiterverarbeitet werden soll.

8.4.15. showDepartureBoard

Zeigt die aktuellen Fahrten der zuvor ausgewählten Station an. Die Anzeige wird live aktualisiert.

Aufbau

function showDepartureBoard(sid, dgid, toTime, rowAnz, tid [, fn])

Parameter

sid: Stations-ID für die Abfahrtstafel

dgid: Daygroup-ID für die Abfahrtstafel

toTime: Zeit in Minuten.als Zahl (Number), als String, als Zeit-String im Format HH:MM oder als Date-Objekt.
Bis zu dieser Zeit wird die Abfahrtstafel angezeigt.

rowAnz: Anzahl angezeigter Abfahrten

tid: enthält die Tabellen-ID für die Anzeige der Abfahrtstafel.

fn: (optional) Callback-Funktion. Diese wird am Ende der Funktion aufgerufen und übernimmt die weitere Verarbeitung.
Wird benötigt, wenn das Ergebnis der Funktion weiterverarbeitet werden soll.