

# FIS - Fahrgastinformationssysteme

## Table Of Contents

- [1 FIS-Gruppe](#)
  - [1.1 Allgemeine und Spezielle FIS-Dateien](#)
  - [1.2 Haltestellen-Liste](#)
  - [1.3 Sonderzeichen](#)
  - [1.4 Routen](#)
  - [1.5 ITCS/RBL-Server](#)
- [2 Anlegen und Bearbeiten von Allgemeinen FIS-Gruppen](#)
  - [2.1 Haltestellen](#)
  - [2.2 Sonderzeichen](#)
  - [2.3 Routen](#)
- [3 Anlegen und Bearbeiten von Speziellen FIS-Dateien](#)
- [4 Script](#)
- [5 Ansagen](#)
  - [5.1 Import der Ansagen](#)
  - [5.2 Anlegen der Speziellen FIS-Datei](#)
  - [5.3 Script](#)
    - [5.3.1 FIS-Gerät](#)
    - [5.3.2 Fahrzeug](#)
- [6 Spezielle Symbole](#)
  - [6.1 Ist in unserer Matrix enthalten](#)
  - [6.2 Ist nicht in unserer Matrix enthalten, aber zwecks Vereinheitlichung reserviert](#)

## 1 FIS-Gruppe

Eine FIS-Gruppe besteht aktuell aus einer Haltestellen-, einer Sonderzeichen- und einer Routen-Liste eines Verkehrsbereiches. FIS-Gruppen werden im ContentTool erstellt (siehe "Zus. Karten-Content"). Mit Verkehrsbereich ist ein in sich abgeschlossenes System von Haltestellen und Routen gemeint - z.B. der Bereich "Tram" der BVG.

### 1.1 Allgemeine und Spezielle FIS-Dateien

Basis und Voraussetzung für jede FIS-Gruppe sind die **allgemeinen FIS-Daten und -Dateien**. Diese umfassen einen Standardsatz von Informationen für jede Haltestelle, jeden Sonderzeichen-Code und den weiteren Daten. Auf diese Weise wird allen Erstellern von Karten, Bordrechnern und Anzeigen ein Standard geboten, sodass - wenn sich jeder daran hält - ohne große Vor- oder Nacharbeiten jedes Fahrzeug auf jeder Karte die dortigen Ziele, Routen usw. nutzen kann, ohne dass es zu Inkompatibilitäten kommt.

Um die Arbeit überschaubar zu halten, sind die Informationen pro Haltestelle, Route usw. aber auf ein Minimum reduziert, sodass Informationen bspw. für Spezialeffekte (invertieren, zusätzliche Daten, Farben usw.) in der Basis-FIS-Datei nicht definiert werden können und sollen. Deshalb gibt es ein dazugehöriges Konzept **spezieller FIS-Daten und -Dateien**. Diese sind einerseits immer mit allgemeinen FIS-Dateien verknüpft und bauen auf diese auf, werden aber auch über eine sogenannte Klasse direkt mit Bordrechner- und Anzeigen-Systemen verknüpft und von denen angesprochen.

Ganz wichtig ist aber, dass die entsprechenden Bordgeräte auch **immer ohne spezielle FIS-Dateien auskommen** und auch die allgemeinen FIS-Daten als Rückfall-Ebene nutzen kann.

Beispiel Nutzung Basis-FIS:



(klick mich, ich bin ein GIF!)

## 1.2 Haltestellen-Liste

In der Haltestellen-Liste befinden sich **sämtliche** Haltestellen, also sowohl die, die als Endhaltestellen mit einem Zielcode versehen sind, als auch die Zwischenhaltestellen, die ausschließlich in den Routen auftauchen.

## 1.3 Sonderzeichen

Damit die Linien-Anzeige auch Linien mit Sonderzeichen oder Symbole anzeigen kann, gibt es in den FIS-Geräten üblicherweise die Möglichkeiten, einen Sonderzeichen-Code einzugeben, der dann die Liniennummer ergänzt, erweitert oder ersetzt. Hierfür enthalten die FIS-Gruppen eine Kodierungstabelle.

## 1.4 Routen

Gäbe es nur die Möglichkeit, eine Linie und eine Zielhaltestelle anzugeben, dann würde das FIS über keine Information verfügen, entlang welcher Haltestellen die Fahrt verläuft, und erst recht nicht, welche Ampel angefordert oder welche Weichen gestellt werden. Auch müsste der Fahrer jede Änderung der Zielanzeige manuell durchführen. Für die Informationen für all diese Funktionen gibt es die sogenannten Routen. Diese bestehen aus zwei Zahlencodes (die Linie und den eigentlichen Routen-Code), einen frei wählbaren String, einer Liste von Haltestellen und optional Angaben über alternative oder sich im Routenverlauf ändernde Ziel-Anzeigen, außerdem über die Angabe einer Folge-Route, die gewählt werden soll, sobald diese Route "abgearbeitet" wurde und einen Sonderzeichen-Code, der automatisch gesetzt werden soll, wenn die Route ausgewählt wird.

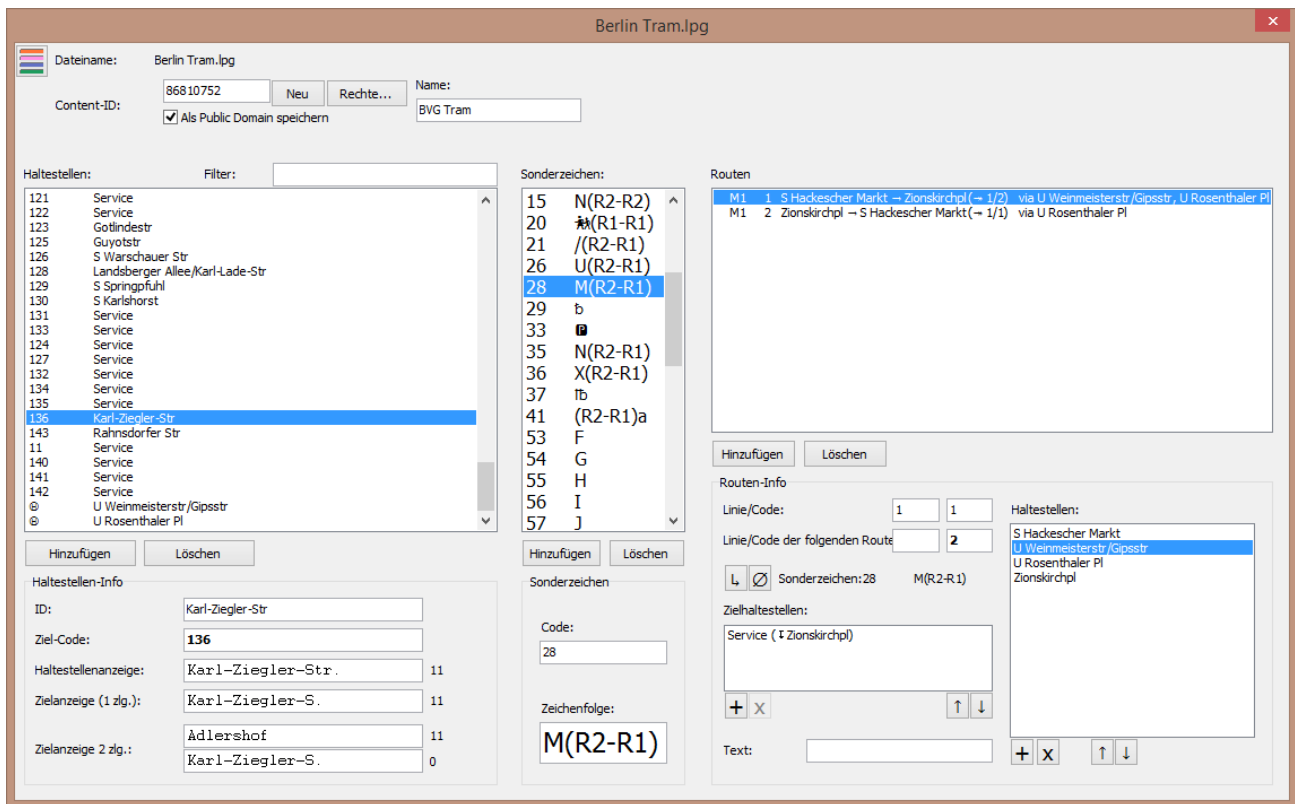
## 1.5 ITCS/RBL-Server

Damit die Zuordnung FIS/RBL-Gerät im Fahrzeug mit den hinterlegten Routen im Fahrplan auch dann klappt, wenn mehrere Verkehrsträger gleichzeitig simuliert werden (oder mehrere Verkehrsgesellschaften), ist es möglich, auf einer Karte mehrere unabhängige RBL/ITCS-Server laufen zu lassen. Um diese eindeutig zu identifizieren, wird sowohl beim Fahrplan, als auch in der Basis-FIS-Gruppe der zuständige Server-Name hinterlegt.

## 2 Anlegen und Bearbeiten von Allgemeinen FIS-Gruppen

FIS-Gruppen - sowohl Basis-Daten als auch spezielle FIS-Gruppen - werden im ContentTool erstellt und bearbeitet. Das zugehörige Tool wird erreicht im Hauptmenü über "Zus. Karten-Content" und dort "FIS-Gruppen (Basis)" oder "FIS-Gruppen (Speziell)".

Die Grundprinzipien des Tools (Laden, Speichern, Packen, ContentIDs usw.) sind dieselben wie bei den anderen Tools.



Ganz links befindet sich die Haltestellenliste - sowohl der Zielhaltestellen als auch der sonstigen Haltestellen. Links steht jeweils der zugewiesene Code, dahinter der unter "ID" angegebene String. Wenn es sich um Zwischenhaltestellen handelt, denen kein Code zugewiesen wurde, dann erscheint vorne statt des Codes ein (H)-Symbol. Wird dagegen ein Code doppelt oder mehrfach vergeben, dann erscheint vor den betreffenden Zeilen ein Doppelkreis. Über der Liste befindet sich ein Suchfeld..

## 2.1 Haltestellen

Wird eine Haltestelle ausgewählt, können unter der Liste deren Daten bearbeitet werden - diese werden hierbei bereits während des Bearbeitens übernommen, es gibt kein OK/Abbrechen o.Ä.. Jede Haltestelle verfügt über folgende Parameter:

- ID: Wird für die interne Verwaltung verwendet, z.B. im MapEditor. Falls es sich um ein Ziel handelt, bei welchem keine Fahrgäste befördert werden sollen, hier bitte "Service" eingeben.
- Ziel-Code: Wird einerseits für den Zugriff des FIS-Gerätes benötigt und entspricht dem realen Ziel-Code. Der Code darf Werte zwischen 0 und 65535 annehmen. Sobald eine illegale Eingabe erfolgt, wird der String durchgestrichen; in diesem Fall einfach korrigieren und die Durchstreichung verschwindet. Handelt es sich um eine Zwischenhaltestelle, dann darf der Code auch weggelassen werden; es wird dann automatisch ein interner Code vergeben. In diesem Fall sollte der Code aber nicht mehr angefasst werden, da bei etwaigen Veränderungen u.U. ein neuer, interner Code vergeben wird und somit bereits erfolgte Zuordnungen in Routen oder speziellen FIS-Gruppen verloren gehen.
- Mehrere Strings, auf die im Script zugegriffen werden kann. Es wird dringend zur Einhaltung der folgenden Konvention empfohlen, damit Basis-FIS-Gruppen und FIS-Geräte jeweils unterschiedlicher Herkunft dennoch gut zueinander passen:
  - Haltestellenanzeige: Dieser String sollte für die Innenanzeigen verwendet werden. Die Innenanzeigen sollen in der Lage sein, mit beliebig langen Strings zu arbeiten (z.B. mit einer automatischen Wechsellanzeige wie in den originalen GT6N-Anzeigen)
  - Zielanzeige einzeilig: Vorgesehen für Außenanzeigen, die nur eine Zeile verarbeiten können. Wir empfehlen eine Begrenzung auf 16 Zeichen, dann ist die FIS-Gruppe sogar zu den üblichen Einzel-Zeichen-Flipdot-Anzeigen der ersten Generation kompatibel, die standardmäßig über 16 Zeichen verfügen
  - Zielanzeige zweizeilig: Wie oben, aber natürlich für zweizeilige Anzeigen. Die beiden Zeilen können vom Script wahlweise zusammen mit Zeilenumbruchszeichen oder einzeln abgerufen werden

## 2.2 Sonderzeichen

In der Mitte befindet sich die Sonderzeichenliste; deren Bearbeitung erfolgt wie bereits bei der Haltestellenliste. Jedes Sonderzeichen besteht aber nur aus einem Zahlen-Code (1 bis 4095) und einem String, der die "Verarbeitung" der Liniennummer beschreibt. Dieser String muss wie folgt aufgebaut werden:

- Buchstaben oder Symbole werden einfach 1:1 angezeigt
- Eingeklammerte Abschnitte werden wie folgt interpretiert:
  - (#) fügt die ganze Liniennummer komplett an dieser Stelle ein
  - (L2-R1) fügt einen Teil der Liniennummer ein, beginnend von (in diesem Fall) dem zweiten Zeichen von links und endend am ersten Zeichen von rechts. "R" und "L" können an beiden Seiten beliebig verwendet werden.
  - \ ( fügt einfach eine "Klammer auf" hinzu (setzt die oberen Regeln außer Kraft)

Beispiel: Aus dem String "M(R2-R1)" und der Liniennummer 123 wird dementsprechend die Anzeige "M23". Wäre die Liniennummer 1234, dann wäre das Resultat "M34". Wäre der String dagegen "M(L2-L3)", dann würde die Linie 123 zwar auch zu "M23" umgewandelt werden, die Nummer 1234 würde aber als "M23" geschildert werden.

## 2.3 Routen

Rechts befindet sich der Bereich zum Definieren von Routen. Die Einträge der Liste geben eine ganz grobe Auskunft darüber, um welche Route es sich handelt: Vorne steht die Linie (mit bereits "eingearbeitetem" Sonderzeichen) und der Routencode, es folgen Start- und Endhaltestelle, ggf. die Angabe der folgenden Route und schließlich die Auflistung der Zwischenhaltestellen.

Jede Route verfügt über zwei Codes zur Zuordnung: Einerseits die zugehörige Liniennummer und andererseits den eigentlichen Routencode (1 bis 65535). Dieser muss wegen der Angabe der Liniennummer deshalb NICHT Liniennummer-übergreifend eindeutig sein! Jede Linie kann also z.B. "ihre" Route 1 und Route 2 bekommen. Die nächste Information ist die der Folgeroute; sofern hier keine Liniennummer angegeben wird, geht LOTUS davon aus, dass die Liniennummer nicht verändert wird. Sobald die Linie/Route-Kombination gefunden wird, werden die Textfelder fett dargestellt, andernfalls kursiv.

Als nächstes gibt es den Sonderzeichen-Code, der automatisch eingestellt werden soll, wenn die Route gewählt wird. Um diesen einzustellen, wählt man in der Sonderzeichenliste in der Mitte das gewünschte Sonderzeichen aus und klickt auf den abgewinkelten Pfeil. Der Code wird daraufhin übernommen und dort dargestellt. Mit dem durchgestrichenen Kreis löscht man das Sonderzeichen. Die Liniennummer wird dann also ganz normal dargestellt.

Dann gibt es die Liste der Haltestellen. Diese muss auch die Abfahrts- und die Zielhaltestelle enthalten. Um hier Haltestellen einzutragen, müssen diese jeweils in der Haltestellenliste ganz links ausgewählt werden und auf "+" geklickt werden. Mit den Pfeil-Schaltflächen kann die Reihenfolge geändert werden.

Links daneben befindet sich die Liste optionaler Zielhaltestellen. Hier gilt folgendes Prinzip: Jeder Eintrag in dieser Liste verfügt über eine alternative Zielhaltestelle und über die Information, ab welcher Haltestelle auf der Route diese gelten soll. Beispiel: Der TXL fährt in Berlin vom Flughafen Tegel zur Memhardstr. am Alexanderplatz und fährt am Hauptbahnhof vorbei. Damit der Durchschnitts-Touri Bescheid weiß, zeigt der Bus bis Hauptbahnhof "Alexanderplatz via Hbf" an und ab Hauptbahnhof nur noch "Alexanderplatz". Bevor er die eigentliche "Haupthaltestelle" vom Alexanderplatz erreicht, schildert er "Memhardstr." und schließlich - unmittelbar bevor er diese erreicht - "Fahrt endet hier" - es handelt sich aber natürlich um nur eine Route. Für diesen Zweck würde die Zielhaltestellenliste wie folgt angelegt werden: Zuerst wird links in der Haltestellenliste eine spezielle Haltestelle, nämlich "Alexanderplatz via Hauptbahnhof" angelegt. Diese muss auch einen eigenen Code haben. Diese wird dann in der Liste ausgewählt. Dann wird auf das "+" unter der Zielhaltestellen-Liste geklickt. Das ausgewählte Ziel wird daraufhin eingetragen und dahinter in Klammern die erste Haltestelle dieser Route (das wäre hier als "Flughafen Tegel"). Das bedeutet, dass das Fahrzeug bereits ab der ersten Haltestelle nicht etwa die eigentliche Zielhaltestelle "Memhardstr." schildern soll (was bei leerer Liste der Fall wäre) sondern eben "Alexanderplatz via Hauptbahnhof". Nun wählt man aus der Haltestellenliste links "Alexanderplatz" aus, klickt wieder auf "+" und klickt auf die Pfeil-Runter-Schaltflächen so oft, bis in den Klammern dahinter "Hauptbahnhof" steht. Dies bedeutet nun, dass das Fahrzeug bis zur letzten Haltestelle vor Hauptbahnhof noch "Alexanderplatz via Hauptbahnhof" schildern soll und - sobald die Haltestelle weitergeschaltet wird - dann das Ziel auf "Alexanderplatz" wechseln soll. Dementsprechend wählt man als nächstes "Memhardstr." aus, fügt diese mit "+" wieder hinzu, verschiebt sie bis 'runter zu "(Alexanderplatz)" und als letztes das Sonderziel "Fahrt endet hier", welches bis hinunter zur "(Memhardstr.)" verschoben wird.

Den Abschluss macht ein Textfeld, welches frei verwendet werden kann, ohne besondere Funktion.

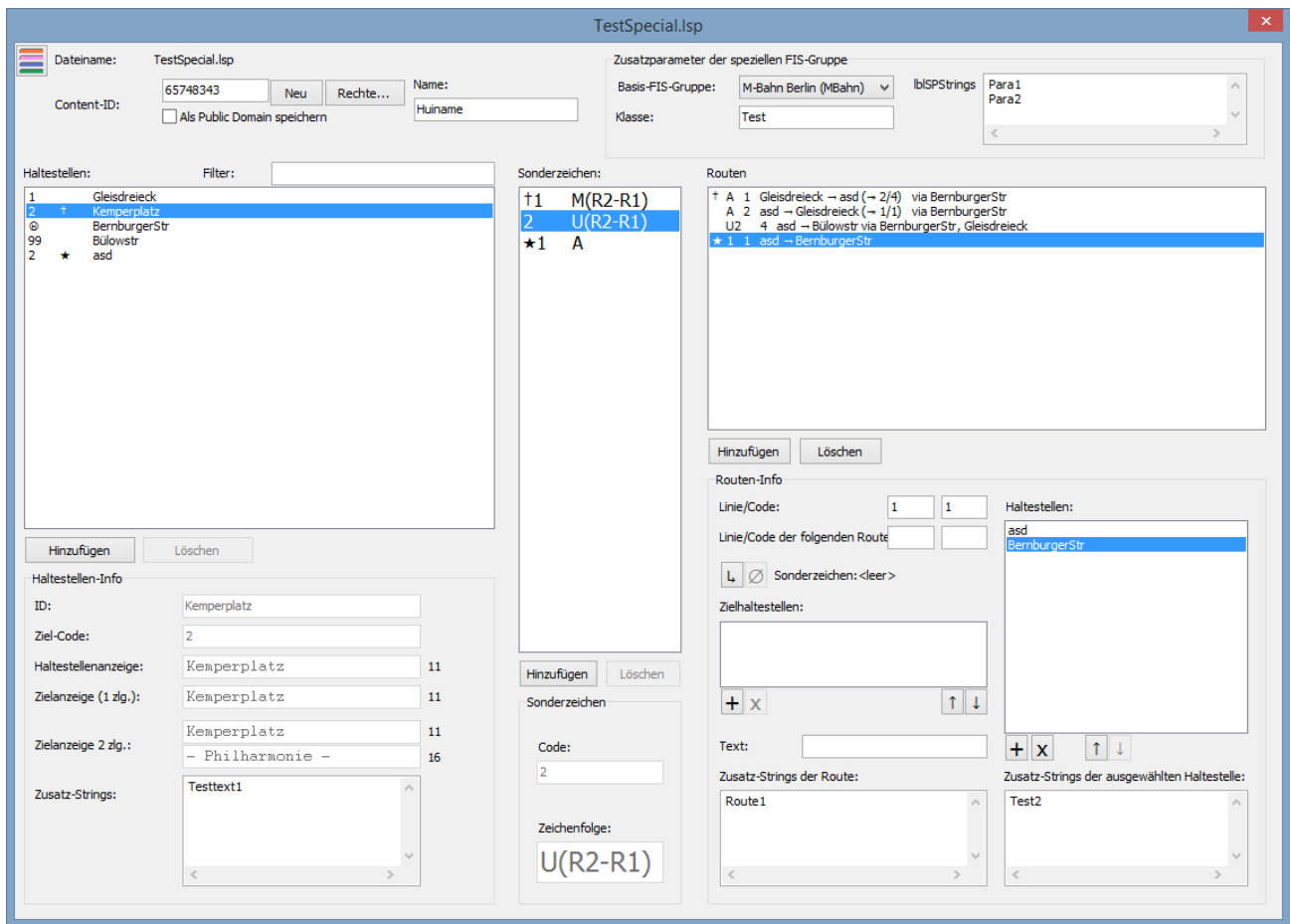
### **3 Anlegen und Bearbeiten von Speziellen FIS-Dateien**

Spezielle FIS-Dateien sind komplett unabhängige Content-Elemente und stellen gewissermaßen eine Ergänzung dar. Jede spezielle FIS-Datei enthält als Parameter die ContentID der zugehörigen Basis-FIS-

Gruppe und einen String, die "Klasse", die das Script verwendet, um auf explizit *diese* FIS-Datei zuzugreifen. Oder anders formuliert: Da das Script bei Angabe einer bestimmten Klasse auch bestimmte Informationen erwartet, sollten dementsprechend Spezial-FIS-Dateien mit derselben Klasse dieselben speziellen Daten enthalten.

Übrigens ist es dem Script möglich, Informationen aus verschiedenen Klassen "gleichzeitig" zu erhalten, sofern also für eine Basis-FIS-Gruppe auch Spezial-FIS-Dateien mit den entsprechenden Klassen vorliegen. Es besteht keine Einschränkung oder Festlegung auf eine bestimmte Klasse.

Das Tool zum Erstellen und Bearbeiten von speziellen FIS-Dateien wird ebenfalls über das ContentTool gestartet, über "Zus. Karten-Content" und dort "FIS-Gruppen (Speziell)".



Der Aufbau des Tools entspricht weitgehend demjenigen für allgemeine FIS-Dateien. Der auffälligste Unterschied ist die Möglichkeit, für jede Haltestelle, jede Route, jede Haltestelle innerhalb jeder Route und für die ganze FIS-Datei selbst beliebig viele Zeilen Text zu ergänzen. Hier lassen sich nun beliebig viele Zusatzinformationen hinterlegen, z.B. alternative Anzeigen, zusätzliche Sonderzeichen, ContentIDs, Marker für Spezialeffekte usw. usw.. Hier wird dann auch ganz deutlich, wofür die Klassen-Information ist: Es kann z.B. festgelegt werden, dass FIS-Dateien einer bestimmten Klasse z.B. in der dritten Zeile immer das Sonderzeichen für XY enthalten. Das bedeutet aber, dass sich alle FIS-Dateien derselben Klasse an diese

Regelung halten und nicht auf ein Mal in der dritten Zeile eine ContentID steht oder so. Jede Klasse sollte also einer festgelegten Konvention folgen, die im Allgemeinen der Programmierer des FIS-Gerätes vorgibt.

Die entsprechenden Texte können einfach so eingegeben werden, wie das mit den anderen Informationen auch erfolgt. Vom Script kann dann unter Angabe der gewünschten Klasse die jeweilige Zeile per Zeilenindex abgefragt werden.

Eine weitere Funktion von speziellen FIS-Dateien ist die Möglichkeit, Haltestellen, Sonderzeichen und Routen zu ergänzen (die funktioniert wie bei den Basis-FIS-Gruppen) oder zu ersetzen. Für letzteres wird auch erstmal eine entsprechende Haltestelle/Sonderzeichen/Route ergänzt und dann derselbe Code vergeben, den die zu ersetzende Einheit trägt. Daraufhin wird in der jeweiligen Liste ein Kreuz vor die zu löschende Einheit gesetzt. Vor jede ersetzende oder komplett neue Einheit wird dagegen ein Stern gesetzt.

## 4 Script

Insbesondere der Bordrechner muss Zugriff auf die Daten der aktuellen [PIS-Group-Datei](#) erhalten, um die entsprechenden Strings auslesen und an die Anzeigen schicken zu können. Hierfür gibt es folgende Funktionen –

Allgemeine FIS-Dateien:

- `PIS_GetStationMainListIndexByTerminusCode(self: integer; code: integer): integer;` Diese Funktion sucht aus der Haltestellen-"Hauptliste" der ausgewählten [PIS-Gruppe](#) die Haltestelle mit dem gegebenen Zielcode `code` heraus.
- `PIS_GetStationStdString(self: integer; stationmainlistindex: integer; stringindex: integer): string;` Diese Funktion gibt einen der in der Standard-Datei hinterlegten Strings zurück. `stationmainlistindex` ist der Index der Haltestelle in der "Hauptliste" (der bspw. mit `PIS_GetStationMainListIndexByTerminusCode` erhalten werden kann) und `stringindex` dient der Auswahl, ob der Stations-String (0) (z.B. Innen-Haltestellenanzeige und Bordrechner), der einzeilige Ziel-String (1), der zweizeilige Ziel-String (2, beide Zeile getrennt durch eine Zeilenschaltung), die erste Zeile der zweizeiligen Zielanzeige (3) oder dessen zweite Zeile (4) zurückgegeben werden soll.
- `PIS_GetStationID(self: integer; stationmainlistindex: integer): string;` Diese Funktion gibt die in der Standard-Datei hinterlegte ID zurück, also den String, mit dem diese Haltestelle in den Fahrplänen identifiziert wird.
- `PIS_GetSpecialChar(self: integer; code: integer; lineinput: string): string;` Diese Funktion generiert aus der Sonderzeichen-Tabelle und unter Verwendung des Codes und dem "Linieninput" die fertige "Sonderzeichen-Liniennummer". Beispiel: Falls für den Sonderzeichen-Code 34 der String "M(R2-R1)" hinterlegt ist, dann würde beispielsweise `PIS_GetSpecialChar(Self, 34, '123')` den String "M23" herausgeben.
- `function PIS_GetRouteIndexByCode(self: integer; line, code: integer): integer;` Diese Funktion sucht aus der Routen-"Hauptliste" die Route mit der gegebenen Linie "line" und dem zugehörigen Routen-Code "code" heraus und übergibt deren Index
- `function PIS_GetRouteString(self: integer; routeindex: integer): string;` Diese Funktion gibt den zur Route gehörenden String aus.
- `function PIS_GetRouteSpecialCharCode(self: integer; routeindex: integer): integer;` Diese Funktion gibt dem zur Route gehörendem Sonderzeichen-Code aus.
- `function PIS_GetRouteStopCount(self: integer; routeindex: integer): integer;` Diese Funktion gibt die Anzahl der Haltestellen dieser Route aus. Die Start- und Zielhaltestelle zählen jeweils dazu.
- `function PIS_GetRouteStopCode(self: integer; routeindex: integer; stopindex: integer): integer;`

Diese Funktion gibt den Haltestellen-Code der Haltestelle an der mit "stopindex" angegebenen Stelle der Haltestellenliste der mit "routeindex" angegebenen Route aus. Achtung: Es handelt sich hierbei wirklich um den Code (!) und nicht um den Index! Den Index (den man z.B. für *PIS\_GetStationStdString* benötigt) erhält man wiederum mit *PIS\_GetStationMainListIndexByTerminusCode*.

- `function PIS_GetRouteTerminusCode(self: integer; routeindex: integer; stopindex: integer): integer;` Diese Funktion gibt den Code derjenigen Zielhaltestelle aus, die an der Haltestelle "stopindex" der Haltestellenliste der Route "routeindex" gültig ist.
- `procedure PIS_GetRouteFollowingCode(self: integer; routeindex: integer; out line: integer; out code: integer);` Die Variablen "line" und "code" werden von dieser Prozedur mit den entsprechenden Codes derjenigen Route beschrieben, die der mit "routeindex" identifizierten Route folgen soll.
- `function PIS_LineRouteCount(self: integer; line: integer): integer;` Mit dieser Funktion kann die Anzahl der zu der angegebenen Linie gehörigen Routen ermittelt werden.
- `function PIS_GetStationCode(self: integer; stationmainlistindex: integer): integer;` Hiermit kann der Code einer gegebenen Station geholt werden. Es ist somit die "Gegenfunktion" zu *PIS\_GetStationMainListIndexByTerminusCode*.
- `function PIS_GetRouteLine(self: integer; routeindex: integer): integer;` gibt die zu der per Index angegebenen Route gehörige Linie zurück.
- `function PIS_GetRouteCode(self: integer; routeindex: integer): integer;` gibt den zu der per Index angegebenen Route gehörigen Code zurück.
- `procedure PIS_GenerateTempRouteListByLine(self: integer; line: integer): integer;` Diese Prozedur legt im Hintergrund eine temporäre Liste sämtlicher zu der gewählten Linie zugehöriger Routen an. Es kann immer nur eine Liste (pro Objekt/Script) gleichzeitig bestehen. Sobald die Prozedur erneut aufgerufen wird, wird die alte Liste gelöscht.
- `procedure PIS_SortTempRouteList(self: integer);` Mit dieser Prozedur wird die temporäre Routenliste nach Code sortiert.
- `function PIS_GetTempRouteListCount(self: integer): integer;` gibt die Anzahl der Einträge der temporären Routenliste zurück.
- `function PIS_GetTempRouteListItemIndex(self: integer; templistindex: integer): integer;` gibt den in der temporären Routenliste an der Stelle "templistindex" hinterlegten Routenindex (nicht Code!) zurück.
- `function PIS_GetITCSServer(self: integer): string;` Diese Funktion gibt den Namen des ITCS- bzw. RBL-Servers zurück. Diese Information benötigt das RBL-Bordgerät, um sich auf dem richtigen RBL-Server anzumelden.

#### Spezielle FIS-Dateien:

- `function PISSP_GetIndexByClass(self: integer; classid: string): integer;` Diese Funktion sucht die spezielle FIS-Datei heraus, die zu der aktuellen Standard-FIS-Datei gehört und zu der Klasse "classid" gehört. Existiert eine solche spezielle FIS-Datei nicht, dann gibt die Prozedur "-1" zurück.
- `procedure PISSP_SetByIndex(self: integer; index: integer);` Diese Prozedur setzt die mit "index" angegebene spezielle FIS-Datei. Fortan berücksichtigen die oben genannten Funktionen und Prozeduren etwaige Änderungen an Haltestellen-, Sonderzeichen- und Routenlisten, die in jener speziellen FIS-Datei vorgenommen wurden. Außerdem wird sie dann für die folgend aufgelisteten Funktionen zugrunde gelegt. "Index" kann mit der vorherigen Funktion "PISSP\_GetIndexByClass" ermittelt werden.
- `function PISSP_GetGroupString(self: integer; stringindex: integer): string;` Diese Funktion liefert die Zeile mit dem Index "stringindex" der allgemeinen zusätzlichen Strings der aktuell gesetzten, speziellen FIS-Datei.
- `function PISSP_GetStationString(self: integer; stncode: integer; stringindex: integer): string;` Diese Funktion liefert die Zeile mit dem Index "stringindex" der allgemeinen zusätzlichen Strings der Haltestelle "stncode" der aktuell gesetzten, speziellen FIS-Datei.
- `function PISSP_GetRouteString(self: integer; routelinecode, routeindex: integer; stringindex: integer): string;`

Diese Funktion liefert die Zeile mit dem Index "stringindex" der allgemeinen zusätzlichen Strings der Route der Linie "routelinecode" mit dem Code "routecode" der aktuell gesetzten, speziellen FIS-Datei.

- `function PISSP_GetRouteStopString(self: integer; routelinecode, routecode: integer; stopindex: integer; stringindex: integer): string;` Diese Funktion liefert die Zeile "stringindex" der allgemeinen zusätzlichen Strings der Haltestelle an der Stelle "stopindex" der Haltestellenliste der Route der Linie "routelinecode" mit dem Code "routecode" der aktuell gesetzten, speziellen FIS-Datei.

## 5 Ansagen

Ansagen werden vom Prinzip her wie folgt realisiert: Die \*.wav-Dateien werden als sogenannte *Unabhängige Sounds* importiert. Hierbei erhält jede von ihnen eine eigene ContentID. Außerdem wird eine *Spezielle FIS-Datei* angelegt, welche die ContentIDs den jeweiligen Haltestellen zuordnet. Das FIS-Gerät-Script holt sich diese aus der speziellen FIS-Datei, verwendet diese, um den zugehörigen Sound auf den Lautsprecher zu setzen und startet dessen Wiedergabe. Im Einzelnen läuft das ganze so ab:

### 5.1 Import der Ansagen

Die Ansagen müssen als \*.wav-Dateien vorliegen und alle, die nachher in einem Container liegen sollen, auch in einem Ordner liegen. Da die alphabetische Reihenfolge für die Vergabe der ContentIDs essentiell ist, sollten sie alle vorne mit einer lückenlosen Abfolge von Zahlen beginnen, die dann den letzten Ziffern der zu vergebenen ContentID entsprechen sollten (inkl. führender Nullen), wie z.B.:

- 01\_Rathaus.wav
- 02\_Schule.wav
- 03\_AusbauWest.wav

Das sind aber alles nur Empfehlungen, keine Bedingungen. Die Reihenfolge hat nichts mit der Reihenfolge der Haltestellen in der FIS-Datei usw. zu tun, sondern lediglich mit der Vergabe der ContentIDs. Welche ContentID nachher welcher Haltestelle entspricht, wird hier noch nicht festgelegt.

Als nächstes muss man ins ContentTool gehen und "Unabh. Sounds" wählen. Es wird nun abgefragt, welcher Ordner importiert werden soll und welche ContentID der alphabetisch erste Sound erhalten soll. Deren letzte Ziffern sollten logischerweise mit denen des ersten Sounds übereinstimmen, also im oberen Beispiel 01. Sollte eine oder mehrere der zu vergebenen ContentIDs bereits vergeben sein (geprüft anhand der vorliegenden Container), dann wird der Import abgebrochen und eine Meldung ausgegeben, welche ContentIDs betroffen sind. **ACHTUNG: Dies bedeutet, dass bei einem erneuten Import von bereits in der Vergangenheit gepackten Sounds, der zugehörige Container zuvor aus dem "MyContent"-Ordner wieder entfernt werden muss!** Andernfalls kommt es nämlich hier zu einem ContentID-Konflikt.

Vor dem eigentlichen Import-Vorgang wird nochmal eine kurze Bestätigung eingefordert. Nach erfolgreichem Abschluss des Imports muss der Container gepackt werden. Hierzu wird einfach das "Container Packen"-Tool im ContentTool verwendet. Nun sollten die importierten Sounds mit ihrer neu vergebenen ContentID abrufbar sein.

### 5.2 Anlegen der Speziellen FIS-Datei

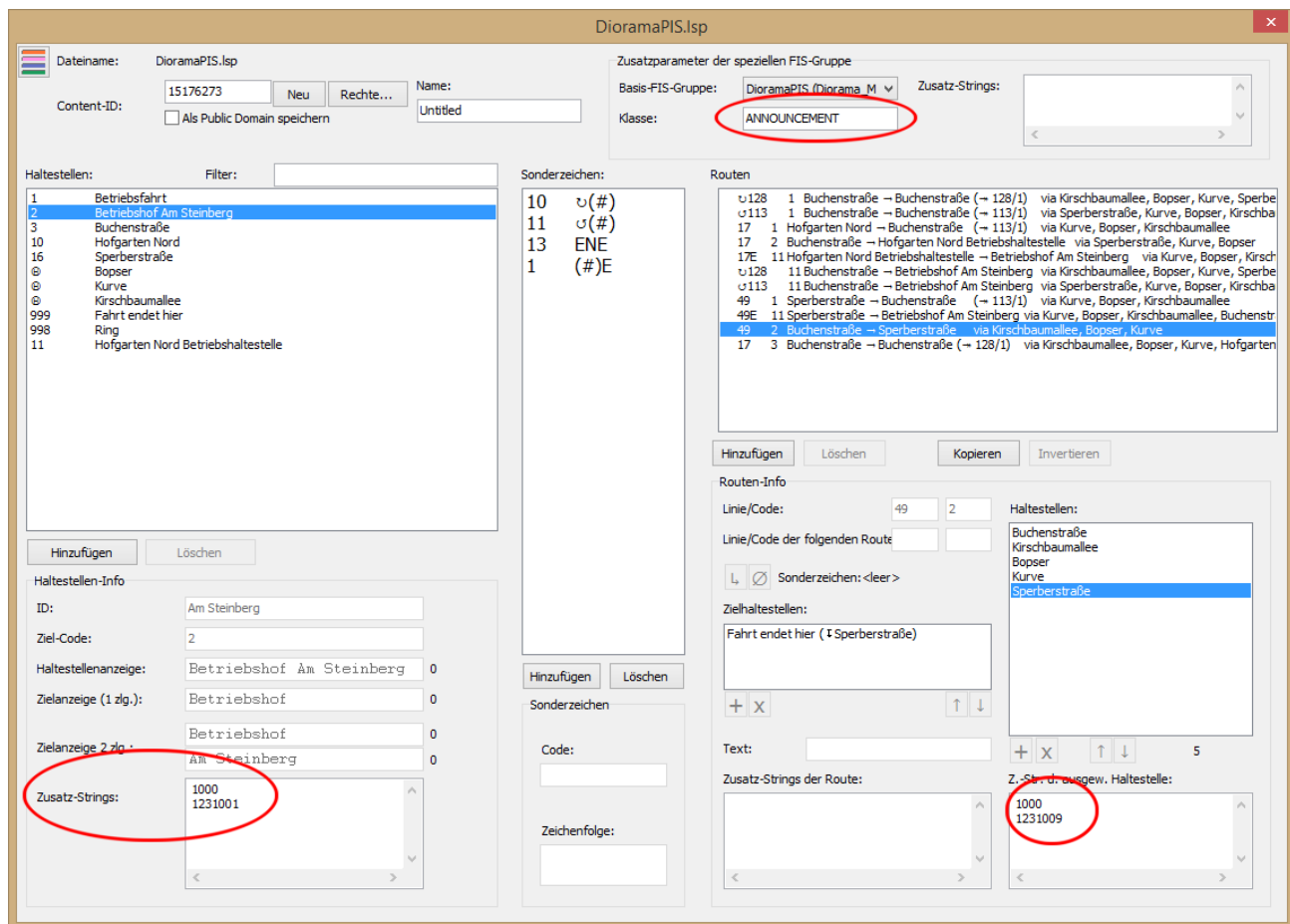
Das Anlegen von Speziellen FIS-Dateien wurde bereits weiter oben erläutert. Möchte man kompatibel zu unseren FIS-Systemen sein, dann muss diese Datei folgender Konvention folgen:

- Klasse: "ANNOUNCEMENT"
- Die Haltestellen der Liste links, die eine Ansage erhalten sollen, werden ausgewählt und unten in den

Zusatz-Strings wird die jeweilige zugehörige ContentID eingetragen, und zwar in der 0. Zeile die UserID und in der 1. Zeile die Content-SubID.

- Für den Fall, dass in einer bestimmten Route an einer bestimmten Haltestelle eine abweichende Ansage abgespielt werden soll (z.B. Endhaltestellen usw.), dann wird dort in gleicher Weise die alternative ContentID in die ersten beiden Zusatz-Strings eingetragen.

Das Ganze sieht dann so aus:



## 5.3 Script

Ich habe das Ansagen-Script beim GT6N auf das IBIS und das Hauptscript verteilt - aus dem Grunde, dass der Lautsprecher sinnvollerweise nicht am IBIS "befestigt" ist, sondern am Fahrzeug selbst. Deshalb gibt es dementsprechend auch zwei Scripts-Ergänzungen:

### 5.3.1 FIS-Gerät

Das FIS-Gerät (z.B. IBIS) macht nichts weiter, als in dem Moment, bei dem die Ansage ertönen soll, die ContentID auszulesen und mittels Broadcast-Befehls an das Fahrzeug weiterzuleiten. Die konkrete Implementierung kann der IBIS\_GT6N.pas im OpenSource-Ordner entnommen werden. Die wichtigsten Punkte sind:

- einmalig den Index der Speziellen FIS-Datei heraussuchen und in eine Variable schreiben:  
`Announcement_PISSP := PISSP_GetIndexByClass(Self, 'ANNOUNCEMENT');`

- beim Aufruf der Ansage zunächst die Spezielle FIS-Datei setzen: `PISSP_SetByIndex(Self, Announcement_PISSP);`
- mit `s1 := PISSP_GetRouteStopString(Self, LinieKurs div 100, Route, StopSeq, 0);` und `s2 := PISSP_GetRouteStopString(Self, LinieKurs div 100, Route, StopSeq, 1);`\* die beiden Strings herausuchen, die ContentID enthalten, für den Fall, dass "rechts" eine ContentID für eine bestimmte Haltestelle einer bestimmten Route eingetragen wurde.
- mit `(s1 <> '')` and `(s2 <> '')` prüfen, ob die Strings auch befüllt sind. Falls nicht, das Ganze noch einmal mit `s1 := PISSP_GetStationString(Self, StopSeqCode, 0);` und `s2 := PISSP_GetStationString(Self, StopSeqCode, 1);`, den ContentIDs der linken, der Haltestellenliste, durchführen und ebenfalls prüfen.
- Die Variablen mit `i1 := strToInt(s1)` und `i2 := strToInt(s2)**` in Zahlen umwandeln
- mit `SendBroadcastInteger(Self, 'PIS', 'ANNOUNCE_USERID', NextAnnouncement_UserID);` und `SendBroadcastInteger(Self, 'PIS', 'ANNOUNCE_SUBID', NextAnnouncement_SubID);`\*\*\* die beiden ContentID-Teile an das Fahrzeug schicken.

\*) Mein IBIS-Gerät verwendet "LinieKurs" so, wie man es auch eintippt, also mit den zwei Kurs-Ziffern hinten angehängt. Aus dem Grunde muss diese Variable vor der Übergabe noch einmal durch 100 geteilt werden, damit sie der in der FIS-Route hinterlegten Liniennummer entspricht.

\*\*\*) Im tatsächlichen IBIS-Script verwendet ich einen sogenannten try-except-Block um die Umwandlungsbefehle, da andernfalls die Gefahr besteht, dass bei ungültigen Eingaben von s1 oder s2 der Scriptablauf hier abgebrochen wird.

\*\*\*\*) Im tatsächlichen Script wird hier stattdessen ein Timer gestartet und deshalb werden diese beiden Befehle erst bei Ablauf des Timers (und dementsprechend an einer ganz anderen Stelle im Script) aufgerufen.

### 5.3.2 Fahrzeug

Für das Fahrzeug wird zunächst eine "Lautsprecher-Soundquelle" erstellt, die nur einen Sound enthält; das kann irgendeiner sein; er wird dann ohnehin im Betrieb ausgetauscht. Diese Soundquelle wird nun im Fahrzeug platziert und ihr ein bestimmter Name zugeteilt, z.B. "Innenlautsprecher".

Das Script im Fahrzeug sieht so aus:

Code

```

1. procedure ReceiveBroadcastInteger(busId: string; id: string; value: integer);
2. begin
3.   if busId <> 'PIS' then
4.     exit;
5.   //..... ggf. andere "Receive"-Abschnitte
6.   if id = 'ANNOUNCE_USERID' then
7.     Announcement_UserID := value
8.   else if id = 'ANNOUNCE_SUBID' then
9.     begin
10.    Snd_Announcement := -1;
11.    if value > 0 then
12.      begin
13.        SndSetInpendant(Self, 'Innenlautsprecher', 0, Announcement_UserID, value);
14.        Snd_Announcement := 1;
15.      end;
16.    end;
17.  //..... ggf. andere "Receive"-Abschnitte

```

22. end;

Display More

Die Variable *Announcement\_UserID* ist global definiert: Der erste Broadcast-Befehl überträgt nämlich die Content-UserID gewissermaßen als "Vorhut" und erst, wenn dann auch die tatsächlich Content-SubID übertragen wird, wird die eigentliche Ansage ausgelöst.

Der alles entscheidende Befehl ist hier `SndSetIndependent(Self, Soundname, Soundindex innerhalb der Soundquelle, ContentUserID, ContentSubID);`, der der genannten Soundquelle den durch die ContentID spezifizierten Sound zuordnet, sodass statt dem "Standardsound" dieser abgespielt wird, sobald *Snd\_Announcement* auf 1 gesetzt wird (die dem Sound ganz klassisch zugeordnete Steuerungs-Variable).

## 6 Spezielle Symbole

Sonderzeichen wie Verkehrsbetriebe-Logos, Stadien, Fußbälle, Flugzeuge usw. werden dadurch realisiert, dass ein Unicode-Zeichen hierfür definiert und in der FIS-Datei hinterlegt wird. Die Schriftarten, die geschildert werden sollen, müssen dementsprechend auch über ein solches Zeichen verfügen, was dort über dasselbe Unicode-Zeichen hinterlegt wird.

Wichtig: Solang man sich nicht im Script befindet, werden in den jeweiligen Textfelder *nicht* die Unicode-Kodierungen *selbst* eingetragene (also dieses U+####), sondern es kann und muss das Zeichen "direkt" eingetragen werden. Am Einfachsten geht das, wenn man es aus der Tabelle unten direkt herauskopiert.

Damit das Ganze einheitlich wird, gibt es hier eine Unicode-Tabelle, an die sich bitte alle Zielerweiterer halten mögen. Selbstverständlich wird sie entsprechend ergänzt, sobald Wünsche geäußert werden! (Gerne gleich mit dem entsprechenden Unicode 😊). Als besonderen Service werden wir unsere GT6N-Flipdot-Anzeige auch immer mal wieder anpassen, sodass die unten aufgeführten Zeichen dann auch bei uns wirklich auf unserem GT6N geschildert werden können.

### 6.1 Ist in unserer Matrix enthalten

Symbolbeschreibung	Unicode-Zeichen	Unicode
Flughafen	?	U+2708
Dreieck	?	U+25B3
Schulkinder	?	U+1F6B8
P+R	?	U+1F17F
DB-Logo	?	U+2114
Fußball	?	U+26BD
Kreispeil Uhrzeigersinn	?	U+21BB
Kreispeil Gegenuhrzeigersinn	?	U+21BA
BVG-Logo	?	U+2422
Brandenburger Tor	?	U+220F
Kreuz	?	U+2715
Umleitungs-Pfeil	?	U+21AC

Symbolbeschreibung	Unicode-Zeichen	Unicode
"U" im Quadrat	?	U+1F687
"S" im Kreis	?	U+1F688
Kaffee-Tasse	?	U+2615
Straßenbahn (von der Seite)	?	U+1F683
Lächelnder Smiley	?	U+1F600
Zwinkernder Smiley	?	U+1F609
Smiley, der die Zunge herausstreckt	?	U+1F61B
(Oriolus-)Vogel	?	U+1F426
Umgedrehtes Pentagramm	?	U+26E7
RhönENERGIE	?	U+0C6F
Werkstattsymbol	?	U+1F527
Bierglas	?	U+1F37A
Ochsenkarren	?	U+1F402

## 6.2 Ist nicht in unserer Matrix enthalten, aber zwecks Vereinheitlichung reserviert

Symbolbeschreibung	Unicode-Zeichen	Unicode
Eingekreiste Buchstaben	? - ?	U+24B6 - U+24CF
Eingekreiste Zahlen	? - ?	U+2460 - U+2473
"Negativ" Buchstaben in ausgefülltem Kreis	? - ?	U+1F150 - U+1F169
"Negativ" Buchstaben in ausgefülltem Quadrat	? - ?	U+1F170 - U+1F189
Nordöstliches Flugzeug	?	U+1F6EA
"BUS" im Kreis	?	U+1F68D
"TRAM" im Quadrat	?	U+1F68A
Hochgestelltes "A"	?	U+1D2C
7/	?	U+2501
5/6	?	U+2502
14/15	?	U+2503
5/13	?	U+2504
5/18	?	U+2505
/ ("Bruch Schrägstrich")	?	U+2044
24/26	?	U+2506
1/20	?	U+2507
2/3	?	U+2508
6/9	?	U+2509
9/18	?	U+250A
22/24	?	U+250B
23/24	?	U+250C
23/26	?	U+250D
25/26	?	U+250E
SB-Logo	?	U+2101
Wiener Linien	?	U+20A9
"S" im Kreis alternativ (z.B. Österreich)	?	U+2A93
"U" im Kreis alternativ (z.B. Österreich)	?	U+FE3C
SEV-Logo	?	U+23E3

<b>Symbolbeschreibung</b>	<b>Unicode-Zeichen</b>	<b>Unicode</b>
Metrobus-Logo	?	U+2133
Bus-Symbol (seitlich)	?	U+1F690
Inventierter Kreis	?	U+25D8
Weihnachtsmann	?	U+1F385
Weihnachtsbaum	?	U+1F384
Vierblättriges Kleeblatt	?	U+1F340
Ausgefülltes, nach oben zeigendes Dreieck	?	U+23F6
Ausgefülltes Quadrat	?	U+2BC0
Pfeil nach oben links neben Pfeil nach unten	?	U+21C5
VDV-Symbol	?	U+2123
DVB-Logo	?	U+2145
Anstoßende Gläser	?	U+1F942
Logo der Dresdener Verkehrsbetriebe	?	U+15EF
Dresdner Silhouette	?	U+1F3D9
Zwingerturn (Dresden)	?	U+2A4E