Selected examples for railML® format (Extract)

RailML version: 2.0-2.2
First issue: July 2010
Elapsed issues: 03.08.2010, 19.03.2012, 26.03.2012, 16.04.2012, 22.05.2012, 16.01.2013
Current issue: 08.12.2014

This document contains an arbitrary selection of typical cases of railway timetables and their indication with railML. This is by far not complete. The possibilities of railML are much more complex than can be shown here.

This is a translation of the corresponding German example selection. Due to normal the gap between the time necessary to translate and the spare time, there are less examples here than in the German version. So, if you miss a certain example which is in the German version, please don't hesitate to contact iRFP, we will try to translate it in a short time.

As far as not named otherwise, the examples here are valid for railML schema versions from 2.0. Some examples refer to attributes which were introduced in versions after 2.0 only. This will be indicated.

Normally the files from which the examples come are available at

www.irfp.de/deutsch/fbs/schnittstelle_railml.html

as railML files as well as PDF files for download.

Theme overview

Theme overview 1
Different stop types 2
Different station names 3
Train types, categories, products, and passenger usage 4
Mileage of tracks and lines 6
On trains and train parts in general 7
Train coupling and sharing 8
Midnight overruns in RailML 11
Header information (Dublin Core Metadata Element Set) 15

Different stop types

General traffic stop

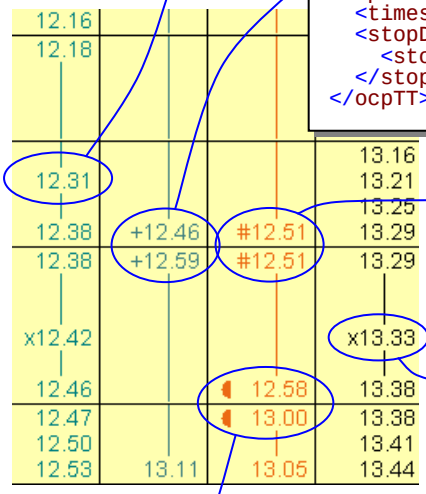
```
<ocpTT ocpRef='ocp_DRAG' ocpType='stop'>
  <times scope='scheduled' arrival='12:30:32' departure='12:31:02'/>
  <stopDescription commercial='true' stopOnRequest='false'>
    <stopTimes minimalTime='PT30S'/>
  </stopDescription>
</ocpTT>
```

Operational stop infrastructure

```
<ocpTT ocpRef='ocp_DKT' ocpType='stop'>
  <times scope='scheduled' arrival='12:51:25' departure='12:51:55'/>
  <stopDescription commercial='false' operationalStopOrdered='false'>
    <stopTimes minimalTime='PT30S'/>
  </stopDescription>
</ocpTT>
```

Operational stop by TOC

```
<ocpTT ocpRef='ocp_DKT' trackInfo='3' ocpType='stop'>
  <times scope='scheduled' arrival='12:45:52' departure='12:59:18'/>
  <stopDescription commercial='false' operationalStopOrdered='true'>
    <stopTimes minimalTime='PT30S'/>
  </stopDescription>
</ocpTT>
```



Stop on request / on demand

```
<ocpTT ocpRef='ocp_DIG' ocpType='stop'>
  <times scope='scheduled' arrival='13:33:26' departure='13:33:56'/>
  <stopDescription commercial='true' stopOnRequest='true'>
    <stopTimes minimalTime='PT12S'/>
  </stopDescription>
</ocpTT>
```

Stop to alight only

```
<ocpTT ocpRef='ocp_DN' ocpType='stop'>
  <times scope='scheduled' arrival='12:58:23' departure='13:00:23'/>
  <stopDescription commercial='true' stopOnRequest='false' onOff='off'>
    <stopTimes minimalTime='PT2M0S'/>
  </stopDescription>
</ocpTT>
```

The attribute stopOnRequest is to be declared only if *commercial=true*.
The attribute operationalStopOrdered[1] is to be declared only if *commercial=false*.

It is not intended to write different stop types at the same station. Concerning the usualities of railway operation:

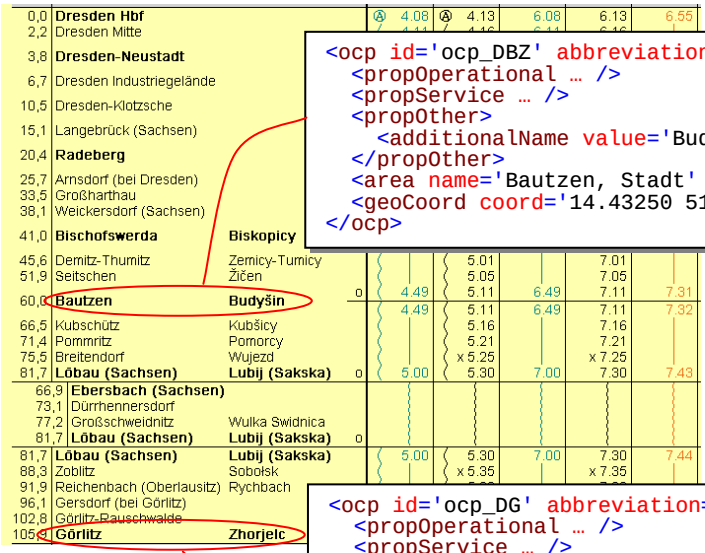
- If there are reasons for both a traffic stop and an operational stop, a traffic stop shall be declared.
- If an operational stop becomes necessary by IM as well as by TOC, it will be declared as an operational stop by TOC (ordered operational stop).

A stop on request is a special case of a traffic stop.

If a stop does not apply to all operating days of the train - i. e. the train runs through at several days - the attribute *operatingPeriodRef* can be used to *reduce* the operating days of the stop against the operating days of the train. However, be aware that other given attributes as run times, supplements etc. become incorrect by this practice. Also, it can only be used to change between a certain stop and run through but not to switch between two different stop types depending on the days of operation. Therefore, many applications would probably split the train into two instead of using *operatingPeriodRef*.

[1] The attribute operationalStopOrdered has been introduced with railML 2.2.

Different station names

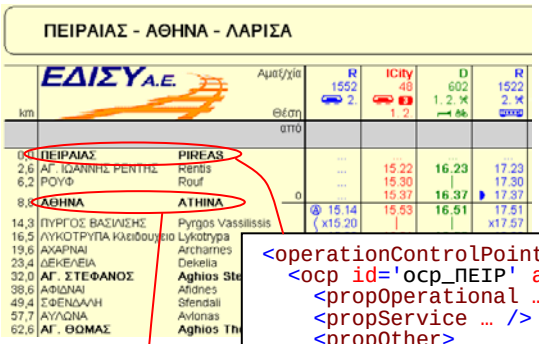



```
<ocp id='ocp_DBZ' abbreviation='DBZ' number='8010026' name='Bautzen'>
  <propOperational … />
  <propService … />
  <propOther>
    <additionalName value='Budyšin' type='trafficName' xml:lang='hsb'/>
  </propOther>
  <area name='Bautzen, Stadt' number='14272010' zip='2625'/>
  <geoCoord coord='14.43250 51.17220 201.47'/>
</ocp>
```

```
<ocp id='ocp_DG' abbreviation='DG' number='8010131' name='Görlitz'>
  <propOperational … />
  <propService … />
  <propOther>
    <additionalName value='Bft. Görlitz Pbf.' type='operationalName'/>
    <additionalName value='Zhorjelc' type='trafficName' xml:lang='pl'/>
  </propOther>
  <area name='Görlitz, Stadt  KfS' number='14263000' zip='2826'/>
  <geoCoord coord='14.98306 51.15072 209.42'/>
</ocp>
```

A different name for operational duties can be addressed with **operationalName**. With **trafficName**, several alternatives for publishing can be addressed. These can differ by language and/or character set.

The attribute xml:lang is optional and seldom used nor necessary. If the attribute is used, the language codes from ISO 639 are to be written.



As the example shows, the difference between the station names can also lie in the character set alone and not in the language. In the example, both names a modern Greek but once with greek and once with latin letters.

```
<operationControlPoints>
  <ocp id='ocp_ΠΕΙΡ' abbreviation='ΠΕΙΡ' name='ΠΕΙΡΑΙΑΣ'>
    <propOperational … />
    <propService … />
    <propOther>
      <additionalName value='Πειραιεύς' type='trafficName' xml:lang='grc'/>
      <additionalName value='ΠΕΙΡΑΙΑΣ' type='trafficName' xml:lang='ell'/>
      <additionalName value='PIREAS' type='trafficName' xml:lang='ell'/>
      <additionalName value='Piraeus' type='trafficName' xml:lang='en'/>
      <additionalName value='Piräus' type='trafficName' xml:lang='de'/>
      <additionalName value='Le Pirée' type='trafficName' xml:lang='fr'/>
```

```
<ocp id='ocp_ΑΘΗΝ' abbreviation='ΑΘΗΝ' name='ΑΘΗΝΑ'>
  <propOperational … />
  <propService … />
  <propOther>
    <additionalName value='Ἀθῆναι' type='trafficName' xml:lang='grc'/>
    <additionalName value='ΑΘΗΝΑ' type='trafficName' xml:lang='ell'/>
    <additionalName value='ATHINA' type='trafficName' xml:lang='ell'/>
    <additionalName value='Athens' type='trafficName' xml:lang='en'/>
    <additionalName value='Athen' type='trafficName' xml:lang='de'/>
    <additionalName value='Athènes' type='trafficName' xml:lang='fr'/>
  </propOther>
</ocp>
```

Train types, categories, products, and passenger usage

Currently there is no attribute at <train> nor <trainPart> where one can directly deduce from whether a train (part) is for passengers or freight. Rather, the attribute categoryRef of the element <trainPart> has to be „traced back“:

```
<trainPart id='tp_222' name='222' line='EC 200' trainNumber='222' processStatus='planned'
                              timetablePeriodRef='ttp_2020_21' categoryRef='cat_EC'>
```

The categories are summarised in an own list at <timetable>. There, the attributes trainUsage and deadRun can be used to determine whether a type of train normally is used for passengers or freight:

```
    <categories>
      <category id='cat_SEV' code='SEV' name='Schienenersatzverkehr'/>
      <category id='cat_OBB' code='OBB' name='Oberlausitz-Bahn' trainUsage='passenger'/>
      <category id='cat_EC' code='EC' name='EuroCity' description='Schnellfahrende Reisezüge
                im internationalen Verkehr mit besonderem Komfort' trainUsage='passenger'/>
      <category id='cat_OBE' code='OBE' name='Oberlausitz-Express' trainUsage='passenger'/>
      <category id='cat_OBC' code='OBC' name='Oberlausitz-City' trainUsage='passenger'/>
      <category id='cat_D' code='D' name='Schnellzug mit Durchgangswagen' description=
                        'Schnellfahrende Reisezüge des Fernverkehrs' trainUsage='passenger'/>
      <category id='cat_Os' code='Os'/>
      <category id='cat_CS' code='CS' name='Ganzzug' trainUsage='goods'/>
      <category id='cat_S' code='S' name='Stadtschnellbahn' description='Reisezüge des
                                linienbezogenen Ballungsverkehrs mit Systemhalten im dichten
                                  Takt unter S-Bahn-Tarifanwendung' trainUsage='passenger'/>
      <category id='cat_FZ' code='FZ' name='FrachtZubringer' trainUsage='goods'/>
      <category id='cat_Lt' code='Lt' name='Leertriebwagen' deadrun='true'/>
      <category id='cat_P' code='P' name='Personenzug' description='Reisezüge des
                                    Binnenverkehrs der ÖBB auf DB-Infrastruktur und Züge der
                                     ÖBB und ČD im Korridorverkehr' trainUsage='passenger'/>
    </categories>
```

It is up to the reading software how it deals with such attributes missing. It can opt for a default case, or ask the user, or force the usage of these attributes e. g. by providing an error message.

The attribute categoryRef escpecially of <trainPart> references to train categories which are commonly known as **products** in practice. They are normally used for publishing. It has to be noted that a train can consist of more than one product at the same time because different train parts may reference different products (see also section *On trains and train parts in general (English)*). This really happens in practice e. g. in Germany between Erfurt and Plaue (Thüringen) where trains of Erfurter Bahn (product *EB*) and of Süd-Thüringen-Bahn (product *STB*) often run coupled, or at ÖBB where sometimes trains with the product name *RailJet* have additional carriages for peak periods placed as *IC*.

In contrary to products there is the operational **train category**. (In UK this is part of the head code of a train at least in a certain sense.) It is something rather internal, not to be published. Also, a train can have only one operational category at the same time. For this kind of train categories, there is the attribute categoryRef at the element <train> in RailML. It is subplaced below <trainPartSequence> because the operational category may change between different sections of the train’s route. The many long-distance trains which run empty before and after their published route are examples from practice for such changes. In Germany, this is widely common between Berlin’s stations Grunewald, Lehrter Bahnhof (main station), and Rummelsburg (stabling station):

```
    <train id='tro_141' type='operational' trainNumber='141' scope='primary'>
      <trainPartSequence sequence='1' categoryRef='cat_IC'>  <!-- IC Schiphol - Berlin -->
        <trainPartRef ref='tp_141_XNSP-BHF' position='1'/>
      </trainPartSequence>
      <trainPartSequence sequence='2' categoryRef='cat_Lr'>   <!-- empty run to stable -->
        <trainPartRef ref='tp_141_BHF-BRGBA' position='1'/>
      </trainPartSequence>
    </train>
```

The attribute categoryRef at <train> for the operational category is normally used at operational trains only (elements <train> with occurrence type='operational').

Currently there is no explicit distinction between *product* and *operational category* in RailML – an element <category> can represent an operational category (i. e. referenced by a <train>) as well as a product (i. e. referenced by a <trainPart>).

Here again it is up to the reading software how to deal with potential contradictions of this contextual redundancy in case a train (part) is shown e. g. as passenger hauling by its <trainPart>.categoryRef and at the same time as non passenger hauling by its <train>.categoryRef. It is recommended to "consult" the product for traffic properties (such as *passenger hauling*) and to "consult" the operational category for operational properties (which may be for instance categoryPriority).

To explicitly declare a train or train part as passenger hauling, it must stringently reference a properly defined <category> by its categoryRef. But, in this relation, it has to be considered that sometimes the so-called **overriding of places** happens. Here, the element <passengerUsage> of <trainPart> is used to override (correct) the place capacities inherited from a formation (by the attribute formationRef). If the place capacities are corrected to 0 by this method, this included implicitly that the train part cannot be passenger-hauling in a certain sense - even if its categoryRef says something else. This method is used e. g. to declare some carriages in a train to be closed. (Some carriages = part of the train may normally be used by passengers but another train part is closed may be because it does not fit at some platform.)

```
<trainPart id='tp_xyz' name='141' … categoryRef='cat_IC'>
   <formationTT formationRef='fmt_abc' … >
     <passengerUsage>
       <places category='class1' count='0'/>
       <places category='class2' count='0'/>
     </passengerUsage>
   </formationTT>
```

Though it alternatively can be expressed by assigning an „empty-run product" to the this train part, it is not always common to do so. Sometimes it is up to the custom of the user and therefore cannot be forced by the writing software. Also, normally there should not be all the train set to 0 places but some of its parts only. Anyway, it is recommended for reading software when determining such properties as *passenger, freight,* or *public* to check any possible overriding of places too and possibly to act according.

Mileage of tracks and lines

Due to historical reasons, the mileage (or "metering") of the tracks of a line often is not continuous. It can have any points of discontinuity ("jump" and/or change of counting direction between raising and falling) for instance by geographical corrections / repositioning of a line.

In railML, the practical, historical mileage (written e.g. at mileposts) is named **absolute mileage**. On the contrary, there is the **relative mileage** (attribute *pos*) which always has to be continuously raising (but not necessarily starting with zero). The relative mileage normally is virtual, i.e. not visible at stations or mileposts. To calculate distances, the relative mileage is used. Note: The term "mileage" here is used in a very general sense and in spite of its measurement unit is defined at kilometers in railML.

A <mileageChange> defines the position of a track where metering of mileage changes. To identify the initial mileage valid from the beginning of a track - before the first mileage change happens - the attributes pos and absPos of the element <trackTopology>.<trackBegin> are used.

The following example contains a track wich
(1) first has a falling mileage starting with km 9.430,
(2) after 9424 meters, so at abs. km 0.006, it jumps to the new value 23.376 and raises from there,
(3) after 18809 meters (from the beginnng of the line), so at the old abs. km 32.761 (= 23376 + 18809 - 9424) it jumps at the new value 33.391 and falls from there.

```
<track id='tr_11.5107_1' name='OBW-OWT' type='mainTrack'>
  <trackTopology>
    <trackBegin id='trn_OBW_11.5107_1' pos='0' absPos='9430'>
      <macroscopicNode ocpRef='ocp_OBW'/>
    </trackBegin>
    <trackEnd id='trn_OWT_11.5107_1' pos='18820' absPos='33380'>
      <macroscopicNode ocpRef='ocp_OWT'/>
    </trackEnd>
    <mileageChanges>
      <mileageChange id='mch_11.07_0' absPos='9430' pos='0' dir='down'/>   (1)
      <mileageChange id='mch_11.07_1' absPosIn='6' absPos='23376' pos='9424' dir='up'/>   (2)
      <mileageChange id='mch_11.07_2' absPosIn='32761' absPos='33391' pos='18809' dir='down'/>   (3)
    </mileageChanges>
    <crossSections>
      <crossSection id='trn_OBW' pos='1290' absPos='8140' ocpRef='ocp_OBW_76A'/>
      <crossSection id='trn_OSML' pos='3570' absPos='5860' ocpRef='ocp_OSML'/>
      <crossSection id='trn_OPT' pos='5580' absPos='3850' ocpRef='ocp_OPT'/>
      <crossSection id='trn_ONKW_H' pos='8930' absPos='500' ocpRef='ocp_ONKW_H'/>
      <crossSection id='trn_ONKW' pos='9430' absPos='23382' ocpRef='ocp_ONKW'/>
      <crossSection id='trn_ONKW' pos='10300' absPos='24252' ocpRef='ocp_ONKW_A'/>
      <crossSection id='trn_ONKO' pos='13460' absPos='27412' ocpRef='ocp_ONKO'/>
      <crossSection id='trn_OWT_N' pos='17945' absPos='31897' ocpRef='ocp_OWT_N'/>
    </crossSections>
  </trackTopology>
</track>
```

Line 1 only exists in FBS-RailML (internal version 2.0.5) vorhanden. Please note how the elements <trackBegin>, <trackEnd>, and <crossSection> contain the current absolute position in their attributes absPos.

The attribute absPosIn specifies the "old" absolute mileage (which is valid up to this mileageChange). This value is redundant so far as it could also be calculated from the previous mileage change.

Remark: The values up and down of this attribute relate to numerical interpretation (raising, falling = “to count up or down”). Therefore, they differ from the typical usage in British English where “up” relates on “direction to London” and “down” relates on “direction away from London”.

For more examples, see
http://www.wiki.railml.org/index.php?title=IS:mileageChange#Example

On trains and train parts in general

One of the base philosophies of version 2.0 of RailML is to satisfy the many requirements of every-day railway operation such as 'strengthen' of trains for raised capacity, direct through-coaches, and even *train coupling and sharing*. This shall be done by "deconstructing" of trains in smallest, atomic fragments. These atomic fragments of trains are called **train parts**.

The actual train information as times, vehicles a. s. o. are properties of the train parts. A <train> structure of RailML only joins train parts to trains ("reconstructs" in the sense of the above named "deconstructing") but, besides this, normally holds no additional information.

While e. g. *operating days* or *no. of vehicles* may change during a train's run, all such properties of a <trainPart> stay constant.

The RailML element <train> can describe either an operational or a commercial train. This is defined by the attribute type which either is *operational* or *commercial*.

The characteristic attribute of **operational trains** is that at one moment there is only one train allowed at a section of line track. This train has clearly to be defined by one "primary key" (called 'head code' or 'train number'). These aspects partly come from reasons of security (as for instance communication between signal boxes).

On the contrary, **commercial trains** are seen from the customers view. They refer to trains as published in public schedules like "Bradshaw's" or modern electronic medias. There may be apparently more than one train simultaneous in one direction of one track of a line. In a time*table* two coupled trains are shown in two separate columns with the same times. In a departure poster there may be two entries with the same departure time and track but bound for different directions (both trains splitting at an intermediate station). Concerning this view, the term "train" is used in a wider sense. So, such a commercial train does not even need to have an engine (e. g. a slip coach).

Each train part normally is used by exactly one operational and one commercial train.

Each train names all its train parts in its element <trainPartRef> with the attributes ref and position. A train may consist of more than one train part either in one section or in subsequent sections of its route. There may be several elements <trainPartRef> with the same position if the corresponding train parts apply in different sections.

Please note that the value of <trainPartRef>.position does not necessarily give the actual position in the train. There may be some train parts "missing" due to different operating days.

More than one train part in one section applies for instance with *train coupling and sharing*. More than one train part in subsequent sections applies for instance if the operating days change at an intermediate station.

Train coupling and sharing

With the term *train coupling and sharing* we describe the situation where two parts of a train (nowadays mostly multiple units) run joined at one section and separated at another section of line.

This principle is not a special case of modern time but well-known throughout the world of railways for a long time. So I will give a few examples only:

- Due to reasons of infrastructure fee and organisation, you'll find many cases of *train coupling and sharing* especially in the today's German regional traffic.
- Not only in Germany, also for instance in the UK's regional traffic we have the trains from Glasgow to Mallaig and Oban running joined to a station called A' Chrìon Làraich… And you'll also find it in Wales but the village names there are even more complicated.
- The same principle applies to the so-called "Kurswagen" or through-coaches so typical of former times. Who knows Agatha Cristie's *Murder on the Orient Express*? The murder took place in the *Calais Coach*, and next to that was the *Athens Coach* which joined the train running from Stamboul to Paris in Belgrade.
- In former times of steam traction in Britain there were *slip coaches* - a special type of direct coach to serve places where the main train was not scheduled to stop.
- Nowadays, we still have through coaches for instance in the overnight trains from Prague and Berlin to Zürich or from Warsaw to Paris and Copenhagen.
- We even find it in countries which are not famous for railway passenger traffic (or, which are even famous for having no railway passenger traffic worth mentioning): The 'Sunset Limited' New Orleans - Los Angeles (train #1) is joined in San Antonio with 'Texas Eagle' Chicago - Los Angeles (train #21) and vice versa.

Example 1

For the sake of readers of UK as well as of the rest of the world (for which *A' Chrìon Làraich* probably means nothing - as *Bischofswerda* of my German examples) let's assume the following example:

A train leaves London St Pancras bound for Paris *and* Bruxelles.
It is split in Lille (capital of French Flanders).

The following RailML extract describes the one **operational train** which starts in London with two train parts, drops off the rear part in Lille and continues its journey with one part only:

```
<train id='tro_9014' type='operational' trainNumber='9014'>
  <trainPartSequence sequence='1'>
    <trainPartRef ref='tp_9014_London-Lille' position='1'/>
    <trainPartRef ref='tp_9114_London-Lille' position='2'/>
  </trainPartSequence>
  <trainPartSequence sequence='2'>
    <trainPartRef ref='tp_9014_Lille-Paris' position='1'/>
  </trainPartSequence>
</train>
```

The following RailML extract describes the other operational train which starts in Lille and runs to Bruxelles:

```
<train id='tro_9114' type='operational' trainNumber='9114'>
  <trainPartSequence sequence='1'>
    <trainPartRef ref='tp_9114_Lille-Bruxelles' position='1'/>
  </trainPartSequence>
</train>
```

So far, we cannot see that the train part which is dropped off the first train in Lille is the same train part which forms the second train. In other words, seeing the operational view only, a

traveller would not know whether he has to change in Lille en route from London to Brussels. Therefore, here come the **commercial trains**.

The following RailML extract describes the through commercial train from London to Brussels:

```
<train id='trc_9114' name='9114' type='commercial'>
  <trainPartSequence sequence='1'>
    <trainPartRef ref='tp_9114_London-Lille' position='2'/>
  </trainPartSequence>
  <trainPartSequence sequence='2'>
    <trainPartRef ref='tp_9114_Lille-Bruxelles' position='1'/>
  </trainPartSequence>
</train>
```

The following RailML extract describes the through commercial train from London to Paris:

```
<train id='trc_9014' name='9014' type='commercial'>
  <trainPartSequence sequence='1'>
    <trainPartRef ref='tp_9014_London-Lille' position='1'/>
  </trainPartSequence>
  <trainPartSequence sequence='2'>
    <trainPartRef ref='tp_9014_Lille-Paris' position='1'/>
  </trainPartSequence>
</train>
```

You may think that `trc_9014` is unnecessary since it holds no new information compared with `tro_9014`. But there should be poetic justice so if the travellers to Bruxelles get their own commercial train… However, please consider that some programmes reading RailML files only look for the commercial trains and do not 'see' any operational train. It also may be a little bit complicated to separate the operational trains which are superset by commercial trains from that which are not. *So, that's why there is the rule that each train part normally has to be used by exactly one operational and one commercial train.*

In general, if we have the situation of *train coupling and sharing*, there is

- one 'long' operational train running at the section where both parts are coupled, and normally, also at one of both 'branches',
- one 'short' operational train running at the other 'branch', so starting or ending at the intermediate station where both train are split or joined,
- two commercial trains (equal before the law), one for each 'branch' but both running through the shared section.

In the example above,

- the 'long' operational train is **tro_9014**,
- the 'short' operational train is **tro_9114**,
- the two pari passu commercial trains are **trc_9014** and **trc_9114**.

(Please note that there is no meaning in the id's. They are just used that way for easier understanding.)

Additionally, you have to have at least four train parts for this example (`tp_9014_London-Lille`, `tp_9114_London-Lille`, `tp_9014_Lille-Paris`, `tp_9114_Lille-Bruxelles`) which are not shown here for shortness. (If you like you'll find extracts from the train parts in the German example above).

As mentioned earlier, there is a difference from the operational and from the commercial point of view. There is only one operational train at the "shared section" (here: London - Lille) but two commercial trains. The train number often given at the column's header of time tables is not identical to the operational train number (or h'code) of a train. This inconsistent usage of the term "train number" is often the reason for misunderstandings: *Does the train between London and Lille has the number 9014 or 9114*?

Example 2

To seek the US-American example again: They use three operational trains for the same situation which makes it even more equal before the law:

```
<train id='tro_1' type='operational' trainNumber='1'>
  <trainPartSequence sequence='1'>
    <trainPartRef ref='tp_01_NewOrleans-SanAntonio' position='1'/>
  </trainPartSequence>
</train>

<train id='tro_421' type='operational' trainNumber='421'>
  <trainPartSequence sequence='1'>
    <trainPartRef ref='tp_21_SanAntonio-LosAngeles' position='1'/>
    <trainPartRef ref='tp_01_SanAntonio-LosAngeles' position='2'/>
  </trainPartSequence>
</train>

<train id='tro_21' type='operational' trainNumber='21'>
  <trainPartSequence sequence='1'>
    <trainPartRef ref='tp_21_Chicago-SanAntonio' position='1'/>
  </trainPartSequence>
</train>
```

Then, we have again the two operational trains. This time I assigned them the name rather than a number since the train number is more typical for the operational view:

```
<train id='trc_SL' name='SUNSET LIMITED' type='commercial'>
  <trainPartSequence sequence='1'>
    <trainPartRef ref='tp_01_NewOrleans-SanAntonio' position='1'/>
  </trainPartSequence>
  <trainPartSequence sequence='2'>
    <trainPartRef ref='tp_01_SanAntonio-LosAngeles' position='2'/>
  </trainPartSequence>
</train>

<train id='trc_TE' name='TEXAS EAGLE' type='commercial'>
  <trainPartSequence sequence='1'>
    <trainPartRef ref='tp_21_Chicago-SanAntonio' position='1'/>
  </trainPartSequence>
  <trainPartSequence sequence='2'>
    <trainPartRef ref='tp_21_SanAntonio-LosAngeles' position='1'/>
  </trainPartSequence>
</train>
```

Well, now imagine: The Sunset Limited takes about 15 hours from New Orleans to San Antonio, leaving New Orleans about noon on Mondays, Wednesdays, and Fridays. The Texas Eagle takes about 30 hours from Chicago to San Antonio, leaving Chicago in the afternoon of Wednesdays and Saturdays. At which position of train #421 is train part #tp_01 from San Antonio to Los Angeles on Thursdays? Do they meet in San Antonio at all?

This shows how difficult real-world examples with train parts may become in conjunction with operating days and midnight-overrun and leads us to the next topics: Operating days and midnight-overrun.

Midnight overruns in RailML

There are several possibilities to indicate midnight overruns correctly in RailML:

- Using the attributes *arrivalDay* and *departureDay*
- By splitting the train's run into <trainPart>s before and after midnight and changing <operatingPeriodRef> between these <trainPart>s…
  a) …and using the attribute dayOffset at the <operatingPeriod> after midnight,
  b) …and using an <operatingPeriod> after midnight which is 'shifted' by one day.

Possibility 1 is the one regularly recommended for midnight overruns during a train run in RailML. It is explicitly not wanted to break a train's run into <trainPart>s *only because of the midnight overrun*. However, possibility 2a comes into consideration for a midnight overrun *before* the current train run (see example below). Possibility 2b shall be avoided.

Midnight overruns inside the current train's route

The attributes *arrivalDay* and *departureDay* are intended for midnight overruns during one train run. They are a counting of the number of midnight overruns relatively to a reference place (e. g. departure, see below). These attributes are optional with default value 0, e. g they do not have to be written as long as a train didn't run over midnight. But, after the first run over midnight, they must be written with values >0.

Example 1: Midnight overruns during an intermediate stop

here midnight overrun

```
<ocpTT ocpRef='ocp_DOLB' ocpType='stop'>
  <times scope='scheduled' arrival='23:59:49' departure='00:00:19' departureDay='1'/>
  <sectionTT section='DOLB-DN_E' lineRef='ln_80_6212' trackRef='tr_80_6212_2' trackInfo='1'>
    <runTimes minimalTime='PT48S' operationalReserve='PT1S'/>
  </sectionTT>
  <stopDescription commercial='true' stopOnRequest='false'>
    <stopTimes minimalTime='PT30S'/>
  </stopDescription>
</ocpTT>
```

All further arrival, departure, and run through times of the train until the end of its route are marked with arrivalDay=1 and departureDay=1.

Example 2: Midnight overrun while in motion

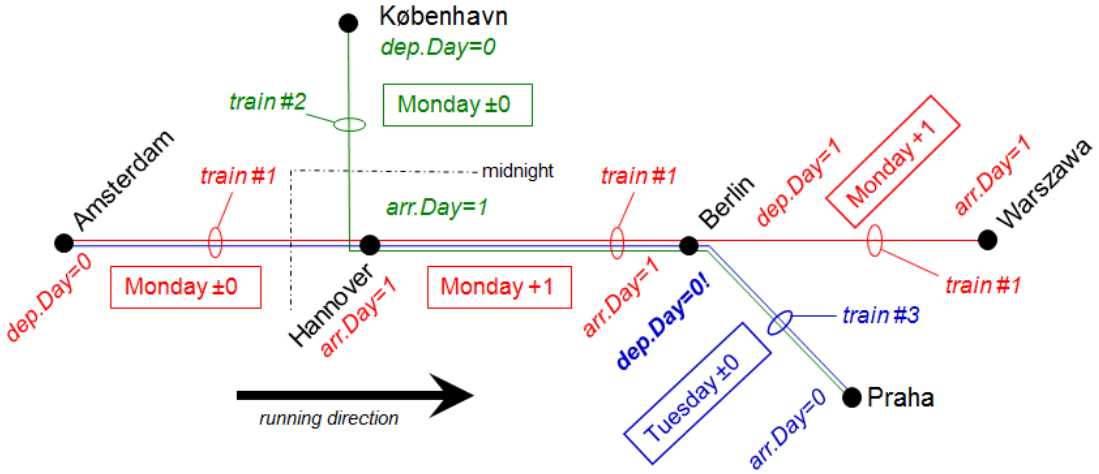
here midnight overrun

```
<ocpTT ocpRef='ocp_DNKW' ocpType='pass'>
  <times scope='scheduled' departure='23:55:00'/>
</ocpTT>
<ocpTT ocpRef='ocp_DNKW_A' ocpType='pass'>
  <times scope='scheduled' departure='23:55:35'/>
</ocpTT>
<ocpTT ocpRef='ocp_DNKO' ocpType='stop'>
  <times scope='scheduled' arrival='23:57:53' departure='23:58:23'/>
</ocpTT>
<ocpTT ocpRef='ocp_DWT_N' ocpType='pass'>
  <times scope='scheduled' departure='00:01:25' departureDay='1'/>
</ocpTT>
<ocpTT ocpRef='ocp_DWT' ocpType='stop'>
  <times scope='scheduled' arrival='00:02:17' arrivalDay='1' departure='00:03:00'
                                                              departureDay='1'/>
</ocpTT>
```

Reference place for day counting / midnight overruns

- It is intended that the arrival/departureDay counting starts with 0 at the first **departure** of a **train**.
- Therefore, the value -1 can occur in rare cases of an arrival before the first departure ("arrival from nowhere", from outside the scope of the RailML file).

- If a <train> consists of several <trainPart>s which are sequentially linked, the day counting normally refers to the first departure **of the whole <train>**. So, it may happen that single <trainParts> already start with day counting >0.
- A "back-jump" of the day counting may happen especially from the view of a commercial train (<train> with type='commercial'): This means that the train first refers to <trainPart>s which did run over midnight and later refers to <trainPart>s which did not run over midnight. See example below.



There are three trains (red, green, and blue) which are 'broken' into at least nine train parts at the black towns. In the run of the green and blue train there is a 'back-jump' of the day index at Berlin (arrDay=1 to depDay=0). This back-jump normally* comes along with an apparent change of operating day (Monday to Tuesday: operatingPeriodRef changes); both phenomena together make it to 'Monday +1' to 'Tuesday +0'.

** 'Normally' as there wouldn't be an apparent change of operating day if the trains would operate daily.*

Do you ask whether there is no such "back-jump" at the Warsaw's route section? There could be. For this example, it is assumed that there is one through operational train from Amsterdam to Warsaw (train #1). Since this begins in Amsterdam before midnight, its day offsets refers to the Amsterdam departure so that the train arrives in Warsaw with arrivalDay=+1 but without back jump. At the Prague branch, in contrast, it is assumed to be a new operational train starting from Berlin (train #3). Its start with departureDay=0 already happens after midnight, hence the back-jump.

<u>Meaning of the day counting / midnight overruns</u>

- The operating day reference (<operatingPeriodRef>) cannot be interpreted alone. This is only possible by including the corresponding day index (arrival/departureDay). To discover at which days a train actually runs it may be necessary to shift the bit mask (from <operatingPeriod>) by the number of bits from (arrival/departureDay + dayIndex).
- Whether a change of operating days during a train run actually happens can only be obtained by comparing the **shifted** bit masks. A change of <operatingPeriodRef> alone is not necessarily a change of the actual operating days.
- There are any numbers of combinations of <operatingPeriod> and arrival/departureDay which are semantically identical, e. g. they effectively describe the same days.
- A writing software is free to choose any combination; it can even change the combination inside one train.

Summary: Combinded example with arrival/departureDay and <dayOffset>

The aim of the following example is to show the two general possibilities of “coaction” of arrival/departureDay and <dayOffset> in context.

Please not that both a and b describe *the same train* in content. Both <trainPart>s sequentially define the train: <trainPart> “tp_1_of_train_1” from A to C, <trainPart> “tp_2_of_train_1” from C to E.

a) …first with one <operatingPeriod> only:

```
<trainPart id='tp_1_of_train_1'>
  <operatingPeriodRef ref='opp_1'/>
  <ocpsTT>
    <ocpTT ocpRef='ocp_A' ocpType='begin'>
      <times scope='scheduled' departure='23:45:18' departureDay='0'/>
    </ocpTT>
    …
!-- here midnight overrun --!
    <ocpTT ocpRef='ocp_C' ocpType='end'>
      <times scope='scheduled' arrival='00:30:40' arrivalDay='1'/>
    </ocpTT>
  </ocpsTT>
</trainPart>

<trainPart id='tp_2_of_train_1'>
  <operatingPeriodRef ref='opp_1'/>
  <ocpsTT>
    <ocpTT ocpRef='ocp_C' ocpType='begin'>
      <times scope='scheduled' departure='00:31:18' departureDay='1'/>
    </ocpTT>
    …
    <ocpTT ocpRef='ocp_E' ocpType='end'>
      <times scope='scheduled' arrival='00:45:40' arrivalDay='1'/>
    </ocpTT>
  </ocpsTT>
</trainPart>

<operatingPeriod id='opp_1' name='Mon-Fri' timetablePeriodRef='…'
        bitMask='0111110011111100…00111110'>
    <operatingDay operatingCode='1111100' />
</operatingPeriod>
```

The midnight overrun is expressed by the attributes arrival/departureDay only. There is only one <operatingPeriod> necessary for this example, the attribute dayOffset is not used.

b) …with two <operatingPeriod>s and the attribute dayOffset:

```
<trainPart id='tp_1_of_train_1'>
  <operatingPeriodRef ref='opp_1'/>
  <ocpsTT>
    <ocpTT ocpRef='ocp_A' ocpType='begin'>
      <times scope='scheduled' departure='23:45:18' departureDay='0'/>
    </ocpTT>
    …
    <ocpTT ocpRef='ocp_C' ocpType='end'>
      <times scope='scheduled' arrival='00:30:40' arrivalDay='1'/>
    </ocpTT>
  </ocpsTT>
</trainPart>

<trainPart id='tp_2_of_train_1'>
  <operatingPeriodRef ref='opp_2'/>
  <ocpsTT>
    <ocpTT ocpRef='ocp_C' ocpType='begin'>
      <times scope='scheduled' departure='00:31:18' departureDay='0'/>
    </ocpTT>
    …
    <ocpTT ocpRef='ocp_E' ocpType='end'>
      <times scope='scheduled' arrival='00:45:40' arrivalDay='0'/>
    </ocpTT>
  </ocpsTT>
</trainPart>

<operatingPeriod id='opp_1' name='Mon-Fri' timetablePeriodRef='…'
        bitMask='0111110011111100…00111110'>
    <operatingDay operatingCode='1111100' />
</operatingPeriod>
```

```
<operatingPeriod id='opp_2' name='Mon-Fri after midnight' timetablePeriodRef='…'
        bitMask='0111110011111100…00111110' dayOffset='1'>
    <operatingDay operatingCode='1111100'/>
</operatingPeriod>
```

The deciding difference is: In example b both <trainPart>s start with `departureDay='0'` whereas in example a the second <trainPart> starts with `departureDay='1'`.

Midnight overruns outside the current train's route

Normally it is intended – for several applications even necessary – that a train starts with arrival/departureDay counting from 0 **from the first departure actually given** (in the RailML data).

Therefore, it is no more possible to use departureDay >0 in case a train did already run over midnight before the first departure (in the RailML data). So, it would only be possible to describe the operating days shifted by the days the train run over midnight before (which is possibility 2b in the list at the beginning).

This solution is explicitly not wanted because of the following reasons (inter alia):

- Timetable periods are normally not 'closed', i.e. after the last day of a timetable period did *not* follow the first day again. A train running daily in a timetable period and running over midnight does not run daily after midnight, strictly speaking: It does not more run at the first day of the period but instead it does run at the first day after the timetable period. The bit mask of its operating days (<operatingPeriod>.bitMask) is shifted by one bit in the direction of raising dates, i.e. even the daily-bitmask containing only 1s before will then start with a 0.
- For several <operatingPeriod>s which are often used for passenger information it may be very difficult to find (understandable) text expressions if they would be shifted by one or more days.

Because of these reasons, the attribute {{TT:Tag|operatingPeriod}}.{{Attr|dayOffset}} has been introduced from RailML 2.2 onwards.

```
<operatingPeriod id='opp_1' name='daily'
       description='runs daily'
       timetablePeriodRef='ttp_2020_21'
       bitMask='111…111'>
  <operatingDay operatingCode='1111111'
              startDate='2020-12-13'
              endDate='2021-12-11'/>
</operatingPeriod>
<operatingPeriod id='opp_2' name='daily +1'
       description='runs daily after midnight'
       timetablePeriodRef='ttp_2020_21'
       bitMask='111…111'
       dayOffset='1'>
  <operatingDay operatingCode='1111111'
              startDate='2020-12-13'
              endDate='2021-12-11'/>
</operatingPeriod>
```

The bit mask of the <operatingPeriod>s with dayOffset≠0 is not to be shifted, i.e. it is identical to the case dayOffset=0.

Basically, there is the possibility (which is principally equal) in such cases to start with arrival/departureDay>0 instead of dayOffset>0 even at the first <ocpTT> of a train. A reading software should be able to parse both versions. The version with dayOffset>0 is intended for cases where departureDay=0 is enforced by external reasons. However, this is also the recommendation of the RailML consortium.

Header information (Dublin Core Metadata Element Set)

The following examples contain a recommendation for the usage of Dublin Core (DC) Metadata Element Set of **rail:metadata** (the head element collection of a RailML file). It is at the moment not necessary to use the DC elements in this way. But since it is not useful for an exchange of data if the same attribute has different meanings it is strongly recommended to use the DC elements in that way.

```
<railml version='2.0' … >
  <metadata>
    <dc:format>2.0.3</dc:format>
    <dc:identifier>1</dc:identifier>
    <dc:language>1252  (ANSI - Lateinisch I)</dc:language>
    <dc:source>iPLAN.exe V1.2.0.528 NtzIntf_RailML2.dll V2.0.4.23</dc:source>
    <!--created with FBS (www.irfp.de) iPLAN.exe V1.2.0.528 NtzIntf_RailML2.dll V2.0.4.23-->
    <dc:date>2012-03-01T11:14:59</dc:date>
    <dc:creator>iRFP</dc:creator>
  </metadata>
```

The attribute **railml.version** shall contain the „Marketing Version“ the scheme files were published with. Typical values are currently **2.0**, **2.1** and **2.2**.

The **Dublin Core Metadata Element Set** (name space „dc:“, structure railml.metadata) is intended for implementation-depending version numbers and other header information.

The element **metadata.format** contains the internal version number of the scheme occurrence (also called RailML *profile*). This version number changes if the interpretation of the scheme by the writing software changes. A reading software should check whether this version number is equal or higher than the version number it was tested with. A reading software also can easily check with this number whether will be some special values which are
- necessary for the reading software,
- optional in RailML,
- obligatory in the specific scheme instance.

It is a matter of the writing software to assign the values for metadata.format. Therefore, these values can only be interpreted correctly together with metadata.source. It is recommended to align them at the official RailML scheme version this implementation is based on. (In the example above: Instance no. #3 of RailML 2.0.) Also it is a matter of the writing software to secure that there are only unique combinations of metadata.format and metadata.source.

The element **metadata.identifier** contains a compatibility number as a simple integer value. This number only changes if the interpretation of an already existing value (attribute or element) changes after its first release (mostly following of an error correction). A reading software should check this value against a certain expected value - otherwise it risks that the data fields do no more contain the expected contents.

For an example, an attribute for speed values could contain speeds in km per hour. This confesses to be not conform to RailML. To provide RailML conformity, this attribute would have to be changed to meters per second - without to be renamed. In this case metadata.identifier would be increased by one to avoid that reading programme which do not know this change read it as kph.

The element metadata.identifier will not be changed if new data are added to a scheme. This is the main difference to metadata.format. It is expected that metadata.identifier changes very seldom compared to metadata.format.

The element **metadata.source** contains a string describing the writing software in a unique matter. Optionally, there may be version numbers to ease error detection.

The element **metadata.language** contains number and (optionally) name of the character set (codepage) the data belongs to. This value is of importance in case the containing Unicode names (station names a.s.o.) have to be converted into a non-Unicode-string by the reading software.

This value is not to be mixed with `<?xml ... encoding='UTF-8' ?>` which defines the coding of the RailML file (and all of its characters). Since a RailML file is normally coded in UTF-8 the value of metadata.language is not necessary for pure reading. It is only necessary if the names have to be converted into a non-Unicode-string for the final target software. A reading algorithm shall not need to 'scan' the names for special characters which would mean a more empirical solution.

For example, in case of `<dc:language>1253 (ANSI - Greek)</dc:language>` a reading programme could - without this statement - only recognise that there are obviously Greek names by recognising strange Unicode-(UTF-8-)values. If a writing programme does not know the origin of the names it shall skip the attribute metadata.language.

metadata.language shall contain as its first characters - until a separating space - the codepage number as decimal numerical value. After that, there may be optionally the name of the codepage.

The element **metadata.date** contains date and time of export (creation of the RailML file) in xs:dateTime format.

The element contains optionally the user name oder licence name of the user or licence used to create the file (e. g. login name of operating system or company name the writing software is licensed to).