Timetable Commands

Overview of all possible timetable commands.

Special fields.

Field in first row and first column must be empty.
Field in first row with #comment and first column with #comment defines timetable reference name.

General definitions

Qualifiers and commands :

$COMMAND [=VALUE] [/PARAMETER[=VALUE]]

A command may have multiple parameters but only one value.

Train reference :

<TRAIN NAME>[:<TIMETABLE REFERENCE>]

The timetable reference must be the timetable reference name.
The timetable reference is not required for trains within the same timetable.
All train names within a single timetable must be unique, but names may be duplicated over multiple timetables.

First column

First column defines fields and stations.

Possible command fields (for full description see below) :

#comment
#consist
#path
#start
#note
#restartdelay
#speed | #speedmph | #speedkph
#dispose

If not a command, the first column is the name of a station or siding.

Empty fields in first column :

- If following a station or siding name, it is a continuation of that row.
- If following a command, it is ignored.

Station qualifiers.

Station name can be followed by the following qualifiers :

- [no]hold
  Hold or do not hold exit signal at danger, overrules the overall route setting.

- Forcehold
  Hold next signal at danger even if this is not recognized as the exit signal for that platform.
  Can be useful to prevent a signal at the next junction to clear too early.

- Forcewait
  Force the train to wait if the next signal is at danger even if this signal is not recognized as the exit signal for that platform.

- [no]waitsignal
  Set to wait or not to wait at platform if signal ahead is still at danger.

- Terminal
  Treat platform as terminal platform, i.e. train will always stop at end of platform.

- Closeupsignal
  Sets a reduced clearance on approach to signal to maximize use of available platform length.

- Extendplatformtosignal
  Sometimes the platform marker is placed some distance from the actual end of the platform where the signal is located, e.g. in case of switches along the platform.
  Normally this would cause trains to stop far from the end of the platform and then block the switches to the rear.
  This parameter will place the 'end of platform' position not at the position of the platform marker but just ahead of the signal position.

- Restrictplatformtosignal
  Sometimes the platform marker is placed beyond the exit signal for that platform.
  If the signal is at danger, the train will stop at the signal and if this is a long train, this stop will not be seen as the station stop as the train has not reached the required platform stop position.
  This parameter will place the 'end of platform' position not at the position of the platform marker but just ahead of the signal position.

- Stoptime = *n* (n is time in seconds).
  Sets the required default stop time at this platform, overriding the stoptime definition set in the tdb.

First row

The first row defines the name of the train. This name must be unique within the timetable.
Possible other values :

#comment : the column holds comment only
$static : the train defined in this column is a nameless static consist.
Note that this train can not be referenced in any command.
empty : continuation of preceeding column, e.g. to hold additional commands.

#Consist

The consist row defines the consist for this train.

NOTE : for trains which are formed through a **$forms** command in the **#dispose** field, or through the **/forms** parameter in a **$detach** command, the consist definition is ignored.
The train will 'inherit' the consist from the train out of which is was formed.

NOTE : consist is also ignored when train is started from a pool.
In this case, the next available train from that pool will form this train, whatever its consist.

NOTE : see below for information on the use of consist information in various commands.

Consist definition :

<CONSIST_NAME> [$REVERSE] [+ <CONSIST_NAME> [$REVERSE] [+ …]]

<consist_name> is the name of the consist file to be use for (part of) this train.
$reverse will use the defined consist in reverse sequence (all cars in the consist will also be flipped in relation to the original car direction).

#Path

The path row defines the path for this train.
All trains need a path, even static trains.

#Start

Defines the start time of this train, in 24 hour clock notation **HH:MM**, or **$STATIC**.

Possible qualifiers when time is set:

- next
  Start time is after 00:00 at the end of the timetable. May be used to start train running after midnight.

- pool = <poolname> [/direction = forward | backward]
  Train originates from the defined pool.
  For trains starting from a pool, the path must start at or near the end of one of the access paths as defined for that pool. If the path starts earlier than the last track section defined for the access path, it must not deviate from that path.
  For turntable pools, the direction in which the train exits from the turntable can be set using the direction qualifier. If not set, the train will reverse.

- create [= time] [/ahead = <train>]
  Normally, a train is created at the defined starttime.
  Using $create, the train will be created as defined in this qualifier. It will be placed as a static train until the defined starttime.
  $Create can only be used for the first run of any train, it is ignored if the train is formed out of another train.

  Value :
  - time
    Time at which the train is to be created.
    If not set, the train will be created at the start of the timetable.

  Parameters :

  - ahead = <train>
    Train will be placed ahead of the train referenced in this parameter.
    Normally, it is not possible to place two trains on the same track.
    Multiple trains can be placed on a single track using this parameter.
    Note that the time set in **$create** must be later as the time as set for the referenced train, except if no time is defined for either train.
    If more than two trains are to be placed on the same track, the third train must reference the second train etc.

- activated
  The train is activated through the **$activate** command from another train.
  The **$activate** command may be send before or after the defined start time of this train.
  A train can be activated by only one other train.

Possible parameters when **$static** is set:

- ahead = <train>
  As above for ahead parameter for the $create qualifier

- pool = <poolname>
  Train is created ***in*** referenced pool. For a pool to have trains, these must be defined using this command.
  The path must be a storage path as defined for that pool. Note that the train may be placed on one of the other storage paths as defined for that pool, this is defined through the pool logic.
  If more trains are created in a pool than the pool can hold, a warning is issued.

#Note

The note row defines commands applicable to when the train is started.

Possible commands :

- acc = <value>
  Sets the required acceleration for this train.
  <value> is a multiplier for the default acceleration.

- dec = <value>
  Sets the required deceleration for this train.
  <value> is a multiplier for the default deceleration.

- doo
  Defines the train as "Driver Only Operated".
  There will be no departure sound (whistle, bell or whatever) on departure from a station.

- forcereversal
  Normally, when a reversal is made and there is a signal in the train's path as leading from the reversal point, the actual reversal position is placed such that the train will be fully passed that signal before reversing, and the reverse move is therefor controlled by that signal.
  Setting **$forcereversal** will allow the train to reverse as soon at it is clear of the reverse position.
  This is useful when shunting in yards when there is no need to fully exit the yard to reverse and the entry signal.

For other possible commands, please see the **Stop Commands** below.

#Restartdelay

Delays are applied when restarting a train from a stop, e.g. at a station or a signal.
Default random delays are set for each train. The default values may be overruled using commands in the #restartdelay field.
The random delay is calculated as <fixed_part> + Random(<variable_part>).

The qualifiers are set as **$NAME [/FIX = <VALUE>] [/VAR = <VALUE>]**
All values are in seconds.

Situation	Default fixed part value	Default variable part value	Command name
New (initial)	0	10	new
Path restriction (signal, reverse switch)	1	10	path
Start when following other train	15	10	follow
Station stop	0	15	station

Special delay for reversing.
When reversing, an additional delay is added reflecting the time required for the driver to walk through or along the train to the other end. This delay is default set to 0.5 sec/m.
It can be overruled using qualifier **$REVERSE /ADDITIONAL = <VALUE>**.
For trains which are pushed on reversal, e.g. for shunt moves of freight trains, it is advisable to set the qualifier **$REVERSE /ADDITIONAL = 0**.

#Speed
#SpeedMpH
#SpeedKpH

This field defines maximum speed for trains, which may restrict the train to lower speed as would otherwise be allowed. Note that any value defined here will never be applied if it exceeds the maximum speed as set through speedposts or signals, or as set in the consist file.
When using **#Speed**, the value is taken as m/sec.
When using **#SpeedMpH**, the value is taken as miles/hour.
When using **#SpeedKpH**, the value is taken as km/hour.

Possible commands :

- max = <value>
  Overall maximum speed for this train.

- cruise = <value>
  Maximum speed at which train will normally operate when it is running on time.
  When the actual delay exceeds the defined maximum delay (as set in **$maxdelay**), the train will accelerate to maximum speed.

- maxdelay = <value>
  Maximum delay (in **minutes**) for cruise control.
  When this delay is exceeded, the train will accelerate to maximum speed.

- creep = <value>
  Creep speed is the minimum speed on the final approach to a signal at danger or station stop location.

- attach = <value>
  Speed at which the train will attach to another train.

#Dispose

The #dispose field defines the action to be taken when the train has reached the end of the path or the final station stop.

Possible commands :

- forms = <train>
  On termination, the train is formed into the indicated new train.
  There will be no change to the train's consist.

- triggers = <train>
  On termination, the train is formed into the indicated new train.
  The train's consist will change into the consist as defined for that new train.

- static
  The train will form into a static train without any further actions.
  There will be no change to the train's consist.

- pool = <poolname> [/direction = forward | backward]
  The train will be taken into the defined pool.
  The path of the train must end on a section which forms part of one of the access paths as defined for that pool. If the path does not end on the first section of the access path it must not conflict with that path, i.e. the path must pass through the first section of the access path.
  For turntable pools, the direction in which the train exits from the turntable can be set using the direction qualifier. If not set, the train will reverse.

- attach = <train>
  The train will attach to the indicated train and will thus cease to exist.

Possible parameters for **$forms**, **$triggers** and **$static** commands :

- closeup
  Final position of train will be close up to end of track or other train.

- setstop
  Time when train terminates is taken as arrival time for new train at station stop.
  Only applies when train forms into new train at station.
  Does not apply to **$static**.

- atstation
  The final position of the train is calculated as if the train is stopping at the station where the new train starts, even if no station stop is defined for this train.
  Does not apply to **$static**.

For possible additional commands in combination with the commands detailed above, see **Stop Commands** below.

Stop Commands

Stop Commands can be placed on any row where a station or siding is defined.
Some commands apply only to an actual stop.
Other commands may be set even if there is no actual stop at that location. In that case, the command applies from that location forward. For instance, a **$wait** command can be set for a station without a stop. The actual wait location can be that station itself, but it could also be a loop or junction somewhere beyond that station.
Some commands can also be set in the **#note** or **#dispose** fields.
If set in the **#note** field, the commands applies from the start of the train or is executed when the train starts.

If set in the **#dispose** field, the command is executed when the train is terminated.
Details for such use are shown in the description for each command.

Commands only applicable to actual stop

All overall commands as defined for the station definitions itself (see details for **First Column** above) can also be set for individual stops.
If the definition in the stop contradicts the definition for that station (e.g. **$hold** is set for the station, and **$nohold** is set for the actual stop), the definition for the stop overrules the definition for the station. For description see definition for stations above.

Other commands for actual stops :

- keepclear
  Defines that the stop position must be such that the length of platform as indicated in the command must be kept clear ahead of or behind the train. This may be essential if another train is to be attached or if another train is to be taken into the same platform.

  Parameters :
  - rear = <n> (n in meter)
    The stop location must be such that he minimal distance behind the train is *n* meter. If the platform has an exit signal, the train will stop in front of the signal even if this means that less than *n* meter is clear, unless the **/force** parameter is set as well.
    In this situation, the path of the train must continue beyond the exit signal.
    Note that the train will never proceed beyond the end of its path.

  - front = <n> (n in meter)
    The stop location must be such that the minimal platform length available ahead of the train is not less than *n* meter. If the rear of the train would be outside the platform, the location is calculated such that the rear of the train is at the platform end even if this means that less than *n* meter is clear, except when the **/force** parameter is set as well.

  - force
    Forces front or rear section to be kept clear even if train must pass exit signal (for **rear** parameter), or rear of train does not fit into platform (for **front** parameter).

- endstop
  When the path of the train continues beyond the station position (e.g. when setting **$keepclear /rear /force**), the stop is considered to be the end of the path even if the train has not reached the actual final position.

- callon
  This train is allowed to proceed into the platform even if that platform is occupied.
  This option requires the TrainHasCallOn or TrainHasCallOn_Restricted function to be implemented for the signal which protects the platform.

- connect = <train>
  This train will wait for the other train to arrive at the station in order to allow passengers to make the connection. Note that a **connect** command will not automatically imply a **wait** command as well. If the train is to follow the other train on the same path, a **$wait** command should be set as well.

  Parameters :
  - maxdelay = n (n in **minutes**)
    This train will only be held for the connection if the known delay of the other train does not exceed the value of **maxdelay**.

  - hold = n (n in **minutes**)
    Time train is to wait after arrival of other train before it can depart. This reflects the time required by passenger off the other train to make the transfer and board this train.
- closeup
  The train will stop close to another train already in the platform.
  Can only be used if callon is also set.

Commands applicable with or without actual stop, for #note or for #dispose field

For each command, it is indicated if this command can also be set for **#note** or **#dispose** field, and possible conditions for such use.

- wait = <train>
  Usage : stop command, #note

  This train must wait for the train as indicated at the first possible location beyond the present location. The wait can apply both to trains in opposite as well as same direction. The paths of this train and the other train will be compared to find the first non-common section ahead of the present position where the wait can be applied.
  Note that any alternative paths defined in either this or the other train's path are not taken into consideration. Care must therefor be taken when using **$wait** commands for trains which also have alternative paths defined, as such a combination may result in a conflict between the **$wait** command and the deadlock processing which could lead to a freeze where both trains are waiting for one another at different locations.

  Parameters :
  - maxdelay = <n> (n in **minutes**)
    The wait is cancelled if the present known delay of the other train exceeds *n* minutes.

  - notstarted
    When a **$wait** command is activated, it is checked if the other train exists. If not, the wait is cancelled.
    If **notstarted** is set and the train does not exist, it is checked if the train still has to

start and if so, the wait is maintained.

- atstart
  The **$wait** is activated at the present position rather than the first non-common position.
  May be used where a train in opposite direction is to terminate in the same location as this train is started and there may not be any possible passing locations between this starting position and the present position of the other train.

- owndelay = <n> (n in **minutes**)
  The **$wait** is only activated if the present delay of this train exceeds the defined value of **owndelay**.
  Note that if the other train is delayed as well, this delay is deducted from the own delay when testing for the **owndelay** threshold.
  For instance, if **owndelay**=10 and the present delay is 12 minutes, the **$wait** would be activated. But if, at that time, the delay of the other train is 8 minutes, the resulting relative delay is only 4 minutes and the **$wait** is cancelled.

- follow = <train>
  Usage : stop command, #note

  Similar as **$wait**, but applies to train in same direction only and applies to full length of path for both trains. This train will therefor never pass the other train even if it could do so where the trains do not have a common path.

  Parameters , for description see **$wait** command :
  - maxdelay
  - notstarted
  - owndelay

- waitany = <path>
  Usage : stop command, #note

  This train will wait for any train which occupies the path as referenced. Note that the defined path and the path of this train must have an overlap otherwise the wait position cannot be located.
  Depending on the parameter, this train will wait for any train on the defined path regardless of its direction, or only for trains in the direction in which the path is defined (default), or in the opposite direction.
  This command can be used on access to single track terminal lines without any loops or sidings to avoid two trains following one another and thereby blocking the return of the first train.
  It can also be used on exit from yard or siding to check on any train approaching on the main line so as not to cause a conflict with such trains.

Parameters :

- both
  Checks for trains in both directions.

- opposite
  Checks only for trains in opposite direction as direction in which path is defined.

If no parameter is set, check is performed for trains in direction in which path is defined.

- detach <detach parameters> <forms parameters>
  Usage : stop command, #note, #dispose (if combined with **$forms**, **$triggers**, **$static** or **$attach**)

  Set details for train to detach a portion of that train.

  Parameters to define the portion to be detached :

  - power
    Will detach the power unit. The system will check for power unit at front or rear, if both are found, front will prevail.
    If there is no power unit at either end, nothing is detached.

  - leadingpower
    Will detach the front power unit only.
    If there is no power unit at the front, nothing is detached.

  - allleadingpower
    Will detach all power units at the front of the train.
    If there are no power units at the front, nothing is detached.

  - trailingpower
    Will detach the power unit which is the rearmost unit on the train.
    If the rear unit is not a power unit, nothing is detached.

  - alltrailingpower
    Will detach all power units from the rear of the train.
    If there are no power units at the rear of the train, nothing is detached.

  - nonpower
    All units which are not power units will be detached from the train.
    The system will determine at which end of the train power units are located, and will then detach all non power units from the other end of the train.
    If neither end has power units, units will be detached from the rear.
    If both ends are power units, nothing is detached.

- units = <n> (n may be <0 or >0 but n=0 is not allowed)
  Number of units to be detached.
  If n>0, the units will be detached at the front of the train.
  If n<0, the units will be detached at the rear of the train.
  If n exceeds the actual length of the train, n is reduced such that one unit remains on the train.

- consist = <consist> [+ <consist> [+…]]
  Name of consist(s) to be detached. For use of consist names in detach command, see note on consist names below.
  The consist to be detached must be at either end of the train, i.e. it must be the front portion or the rear portion of the train.
  If a list of consists is defined, it must be in the sequence of the consists to be detached, from the outside looking inward, i.e. if the units are to be detached at the front, the first consist in the list must be the front portion, but if the units are to be detached at the rear the first consist in the list must be the rear portion.
  If neither front nor rear portion matches the consist or first consist as defined, nothing is detached.

Parameters for formed train :

- forms = <train>
  Detached portion will form train as indicated.

- static
  Detached portion will form a static consist.

- attach = <train>
  Usage : stop command, #dispose (may not be combined with other command except **$detach**)

  This train will attach to train as indicated, and will therefor cease to exist.
  If used at station stop, there is no use to define anything beyond this stop, and nothing can be defined in the **#dispose** field either.
  If the other train to which this train must attach is not at the location where the attach is to take place, this train will terminate without the attach taking place.
  It is therefor advisable to use a **$wait** command to ensure the other train is in the location as required.
  If the **firstin** or **setback** parameter is set, it should be the other way round, in that case a **$wait** command should be set for the other train to ensure this train is indeed first in.

  Parameters (only valid at station stop):

- firstin
  This train is in first, and will wait for arrival of the second train to perform the attach. The other train may come in ahead of this train through a switch or from the opposite direction.

- setback
  This train is in first, and will wait for the other train to come in behind. When the other train has arrived, this train will set back to perform the attach.
  This should not be used if an engine is to be detached from the other train as this train will not wait for the engine to clear before performing the attach.

- pickup [=<train>] [/static]
  Usage : stop command, #dispose (combined with $forms, $triggers only)

  This train will pick up the train as defined in the command, or will pick up the static consist which is on the location where the pickup is defined.
  The train which is picked up will cease to exist. The full train is picked up, no changes are made to the consists of either trains (except if combined with $triggers command in #dispose field).
  If there is no train to pick up at the required location, the train will continue as defined.

- transfer [=<train>] [/static] <transfer parameters>
  Usage : stop command, #dispose (combined with $forms only)

  This train (the 'active' train) will transfer units with the train as indicated, or with a static consist placed at the location where the transfer is defined (the 'passive' train).
  With a transfer, units will be transferred from one train to another, but both trains will continue to exist.
  At least one power unit must remain on the 'active' train, this power unit must not be part of the portion to be transferred.
  The 'passive' train need not have power units, or all power units may be detached as part of the transfer.

  Parameters defining the type of transfer :
  - give
    This train is to give the defined units to the other train, that is units as defined for the 'active' train will be moved to the 'passive' train.

  - take
    This train is to take the defined units from the other train, that is units as defined for the 'passive' train will be moved to the 'active' train.

  - keep
    All units except the units as defined for the 'active' train will be transferred to the 'passive' train.

- leave
  All units except the units as defined for the ‘passive’ train will be transferred to the ‘active’ train.

Parameters defining the units to transfer or to keep on the train :

- onepower
  One power unit only.

- allpower
  All power units.

- nonpower
  All units which are not power units.

- units = <n>
  If the portion is defined for the ‘active’ train, and <n> exceeds the length of that train, the number is reduced such that one unit will remain on the train.

- consist = <consist> [+ consist [+…]]
  Consists names of portions to keep or to transfer.
  The consist names must be in sequence, and the first (or only) consist name must match the portion at the applicable end of the train.

- activate = <train>
  Usage : stop command (with actual stop only), #note or #dispose

  Will activate the train as indicated, either when the train starts, when the train is at the indicated stop or when it is terminated.

Notes on $detach, $attach, $pickup and $transfer commands

- If in a **$detach** command, all driveable power units remains on the present train, the player will continue with this train.
  If in a **$detach** command, all driveable power units are detached and form part of the new train, the player will automatically continue with the new train.
  If in a **$detach** command, both continuing and new train have driveable power units, a window will pop up informing the player of the intending detach, and in which portion the present player locomotive is located. The player can switch to another power unit in the other portion using the normal ‘switch cab’ command.
  When the player is in the portion as required, the detach can be confirmed by the player and will only then take place.

- In an **$attach** command, the train performing the attach will cease to exist, the other train will continue.
  If the train performing the attach is the player train, the player will automatically continue with the train to which the train was attached.

- In a **$pickup** command, the train which is picked up will cease to exist.
  If the train which is picked up is the player train, the player will automatically continue with the train which performs the pickup.

- In a **$transfer** command, both trains will continue to exist.
  If either of the trains is the player train and the transfer includes all driveable power units, the player will continue with the other train.
  If either of the trains is the player train and the transfers defines that some but not all driveable power units are detached from the present player train, this is considered as a **$detach** command with both portions containing driveable power units, and the procedure is the same as for the **$detach** command as described above.

- If power is to be detached in a **$detach** or $**transfer** command, the following will apply :
  - If a single power unit is to be detached, and the first unit is an engine, any and all following units of type tender will also be detached.
  - If a single unit is to be detached, and the first unit is a tender, any and all following units of type tender will also be detached, up to and including the first unit of type engine.
  - If all power is to be detached or retained, any unit of type tender is assumed to be a power unit.
  - No checks on type of unit are performed if **consist** or **unit** is defined as transfer portion.

- A train to which another train is to attach as defined in a **$attach** command, or a train which is set as the 'passive' train in a **$transfer** command, will be held at the location where the attach or transfer is to take place until the attach or transfer has indeed taken place.

Notes on the use of Consist information in $detach and $transfer commands

Any wagon on the simulation must have been placed somewhere as a 'new' train. When a 'new' train is placed, it is formed as defined in the consist definition for that train.
Each wagon will remember this 'original consist' throughout its entire life on the simulation.
This 'original consist' name can be used in any $detach or $transfer command, even if the portion involved has changed trains.
So, for instance, if a freight train is placed which consists of multiple portions, each with their own consist name (using the multiple consist definition), each wagon in that train will always remember its original consist. When this train is taken apart, portions are taken into other trains etc., the original consist name can still be used.
When using this facility it is important to keep track of where and in which train the various portions are moved. As a list of consists must be defined in the correct sequence, it is also important to keep

track of the configuration of the formed trains.
The advantage of this method is that one does not need to keep count of the number of units in each train and each portion.

Note that the consist information can not be used if the unit is started at a pool, if that pool can hold different consists. In that situation, it is not defined which consist will form the actual train.

Pools

Introduction

Pools can be used to store trains before or in between active duties, or when all duties have been performed.
Trains can be defined to be placed in a pool at the start of the timetable. When required, the train can be extracted from the pool. When the duty has terminated, the train can be returned to the pool.
There is no need to define the exact storage of the train, nor is there need to sort out the various duties so as to avoid trains being locked in by other trains which are only required at a later time. When using pools, the system will take care of actual storage location and will select the first available train when a train is required.
A pool will consist of one or more tracks which are used to stable the trains. Access tracks must also be defined. For details, see below.

A special type of pool is the turntable pool. In a turntable pool, all storage tracks are connected to a turntable. The access paths are also connected to the turntable. When extracting or storing a train, the train will run unto the turntable and the turntable, with the train on it, will be turned to the required position.

Pools can be used for both AI and player train.
When a train which is extracted from a pool is selected as the player train, the first available train will be selected and set as player train.
When a train which is the player train is send to a pool, the train will terminate in the pool. The player can remain with the train until its next duty, but there is no way to tell what or when that duty will be, as that depends on other actions set up for that pool.

Conditions and restrictions

General restrictions and conditions which apply to normal and turntable pools.

- A pool can only contain trains which are equivalent in usage.
  The trains need not all be same type, but their use must be exchangeable.
  It is not possible to select a specific type of train from a pool.

- Attach, detach or transfer is not possible for trains stored in a pool.
  Only fixed formations (single or multiple engines, or MU's) can be extracted from or send to a pool.
  If multiple units are required, these must be extracted separately and coupled together after exiting from the pool.
  If multiple units are to be send to a pool these must be detached before send to the pool.
  As attach, detach or transfers are not possible, pools can only be used by engines and MU's, i.e. for units which can move on their own. Pools can not be used for coaches and wagons or trains without power.

Pool underflow and overflow

Pool 'overflow' can occur when a train is send to a pool but the storage area is full to capacity.
In this situation, the train will terminate at the access point to the pool, and will be removed.
Pool 'underflow' can occur when a train is requested from a pool but the storage area is empty and no units are available. In this situation, if the flag 'force creation' is set for this pool, the train will be created and will start at the access point. If this flag is not set, the train is cancelled.
A warning is issued to the logfile in case of pool underflow.

Pool definition

Pools are defined in a file similar to a timetable file, i.e. spreadsheet saved as unicodetext file.
The files must be stored in the same directory as the timetable files (***<route> \Activities\OpenRail***).

Field in first row and first column must be empty.
Other fields in first row are not used.
All other fields in the first column contain the parameter definition, the fields in the second column contain the parameter values. For details see below.
The file extension for normal pools is **.pool-or**, for turntable pools this is **.turntable-or**.
A file can hold the definition for one or more pools, these need not be related in any way.

Definitions for normal pools.

Parameters :

- #comment
  Comment only, value is ignored.

- #name
  Name of the pool. This is the name which must be used in the timetable $pool commands for creating, extracting or storing trains for this pool.
  This field is compulsory, and must precede all other parameters.

- #storage
  Path which defines a storage track.
  The path must be defined in **outbound** direction, that is the direction of the train when it leaves the pool.
  A storage path can only be a single section, it can not pass over switches or crossings.
  At least one storage track must be defined.

- #access
  Path which defines access to a storage track.
  The path must be defined in **outbound** direction, that is the direction of the train when it leaves the pool.
  The path can pass over switches or crossings but can not contain any reversal points.
  Each storage track definition must be followed by one or more access path definitions.

- #maxunits
  For each storage track, the maximum number of units which can be stored on that track can

be defined.
This field is optional.
Note that this defines only the maximum number of units, the actual number may be less if the length of the storage track is not sufficient to hold this number of units.

- #settings
  Special flags for pool usage.
  Only one value is allowed :
  - force creation
    Trains are created on access point in case of pool underflow.
    This flag applies to the full pool, and need only be set once per pool.

Notes :

- Access paths to storage tracks can only be defined at one end of the storage track. Trains will always enter and exit the pool at the same end.
  It is not possible to define 'run through' storage areas.

- Although each storage path has its own access path(s), it is advisable that all access paths end at the same point, such that all storage tracks are accessible from that location.
  It is possible to have multiple access points but then it is still advisable that all storage paths can be reached from all points.
  If only part of the storage paths can be accessed from an access point, there is a risk that the trains can not be spread adequately over the full storage area. Worst case, if all trains are always send to one access point and always extracted from another access point and these points do not access all storage tracks, there may be a continuous series of pool 'overflow' and 'underflow' as the engines send to the pool can not be extracted.

Definition for Turntable Pools

A turntable pool is special type of pool based on a working turntable.
All storage and access tracks must be linked to the turntable.
As different types of trains can use the turntable, a turntable can be part of multiple pools as different pools are required for each different type of train.
Storage tracks can not be shared by different pools, each pool must have its own set of tracks.
Access paths can be shared by multiple pools.

Parameters :

- #comment
  Comment only, value is ignored.

- #name
  Name of the pool. This is the name which must be used in the timetable $pool commands for creating, extracting or storing trains for this pool.

This field is compulsory, and must precede all other parameters.

- #worldfile
  The filename of the worldfile in which the turntable is located.

- #uid
  The uid of the turntable in the worldfile.
  Together with #worldfile, this defines the turntable on which the pool is based.
  The #worldfile and #uid values must be the same as the related values in the turntable.dat file which defines the working timetables.

- #storage
  Path which defines a storage track.
  The path must be defined in the direction **leading away from the turntable**.
  The start position of the path must be outside the turntable area.
  A storage path can only be a single section, it can not pass over switches or crossings.
  At least one storage track must be defined.

- #access
  Path which defines access to a storage track.
  The path must be defined in the direction **leading away from the turntable**.
  The start position of the path must be outside the turntable area.
  The path can pass over switches or crossings but can not contain any reversal points.
  At least one access path must be defined. The access path is not linked to a specific storage track but applies to all storage tracks as these are always accessed via the turntable.
  See note below on use of access paths.

- #maxunits
  For each storage track, the maximum number of units which can be stored on that track can be defined.
  This field is optional.
  Note that this defines only the maximum number of units, the actual number may be less if the length of the storage track is not sufficient to hold this number of units.

- #speedmph
- #speedkph
  These parameters define the maximum speed of train when accessing the turntable (in mph or kph). There is no need to place speedposts in the route to limit the speed on the turntable.
  The maximum speed as defined for the turntable will also apply to the storage tracks.
  On exiting on the turntable on access paths, the train will automatically revert to the maximum speed which applied on the approach to the turntable.

- #framerate
  This parameter defines the framerate for turning the turntable.

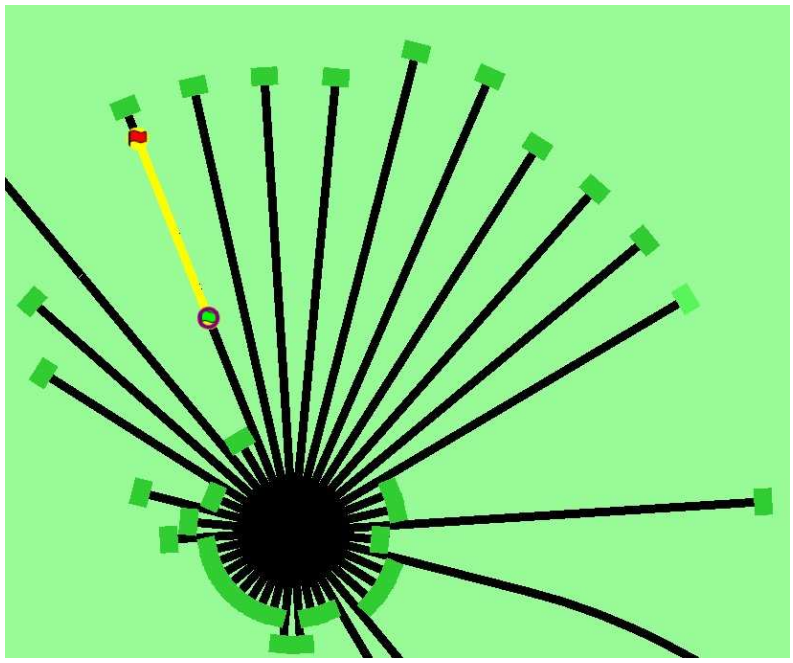
See below for full description of the use of this parameter.

- #settings
  Special flags for pool usage.
  Only one value is allowed :
  - force creation
    Trains are created on access point in case of pool underflow.
    This flag applies to the full pool, and need only be set once per pool.

Notes :

- Note the following differences between normal pools and turntable pools :
  - For normal pools, each storage path must have at least one access path;
    for turntable pools, access paths are independent from the storage paths.
  - For normal pools, storage paths are defined in **outbound** direction;
    for turntable pools, storage paths are defined leading away from the turntable so in **inbound** direction.

- The trackviewer will show paths leading through the turntable.
  The definition of turntable paths, however, must not pass through the actual turntable itself, but must start outside the turntable area, as shown below.



- It is advisable to have separate access paths for extracting trains from the pool and sending trains to the pool, especially if the turntable is shared by multiple pools.
  Otherwise, if a train is send to the turntable at approximately the same time as another is extracted, there is a risk of a deadlock situation. The program cannot resolve this as it cannot see that the trains are bound to proceed onto the same track while the train which is

extracted is still waiting for the turntable or is being turned.

- The turntable will always move to the required position over the shortest angle.

- When a train requests the turntable but the turntable is already activated or occupied by another train, the request is queued.

- After the engine has been turned and moves off the turntable, the turntable is released when the engine is a short distance clear of it. If no other requests are queued, the turntable will remain in that position until the next request.

- Turntable framerate.
  Normally, the turntable framerate (i.e. the speed at which the table rotates) is taken from the shape file of the turntable.
  However, as AI trains can use a turntable anywhere on a route, it may be that the shape file of a particular turntable which is not in view has not been loaded, and therefor the framerate can not be derived in that way. The value as defined for the pool is used as substitute.
  If at any time the turntable is used when its shape file is loaded, this substitute value is replaced by the value as defined in the shape file.
  One frame per second relates to a rotation speed of 0.1 degree per second.
  This parameter is optional, if not defined, a default value of 30 frames per second is used, which gives a default rotation speed of 3 degrees per second.

- AI train behaviour on accessing the turntable :
  - When an AI train approaches the turntable on an access path and the turntable is not in the required position, the train will stop just short of the turntable and will request the turntable to move to that position.

  - When an AI train is requested to exit from a storage track and the turntable is not in that position, it will request the turntable to move but will not start to move toward the turntable until the turntable is in position.

- Player train behaviour on accessing the turntable :
  - When the player train approaches the turntable on an access path and the turntable is not in the required position, stop just short of the turntable and wait until the table is in position. There will be a screen notification when the turntable is ready.

  - When the player train is extracted from the pool, the turntable will turn to the required position. The player train can either wait or move slowly toward the turntable. There will be a screen notification when the turntable is ready.

  - When moving onto the turntable, proceed until the engine is fully positioned on the turntable. There will be a screen notification when the engine is correctly positioned.

- When the engine is positioned, set the throttle to 0, and set the reverser to neutral (or 0% for steam engines).
  The turntable will start to move when both conditions are met. Do not move the engine while the table is turning.

- When the turntable is in the required position, the train can be moved off the table.

- **Do not at any time move the turntable using manual controls.**