1. Introduction.

This document is intended as preliminary documentation for the changes in timetable mode as committed in May 2017.

2. Changes to file format.

In order to properly process non-standard characters in station names, the file must be stored as unicode .txt file with ‘tab’ as separating character.
For routes which do not hace station names which contain non-standard characters, files stored as .csv with ’:’ or ’:’ as separator will still be supported.
The proper file-extention is timetable-or, but timetable_or will also still be supported.

3. Coupling and Uncoupling of trains.

3.1. Notes.

3.1.1. General notes.

- When coupling or uncoupling to or from the player train, it is not required that the player operates the angle cocks and brake hoses, this is performed automatically.
- The player train cannot just couple to any other train, coupling is only possible if this is defined in the timetable.
- Uncoupling from the player train cannot be performed using the Train Operations Window. If uncoupling is defined in the timetable, it will be handled automatically when the train stops at the required location. It does not require any actions by the player.

3.1.2. Notes on power units.

Uncouple commands allow the use of ‘power unit’ definitions.
The following applies when ‘tenders’ are included in the train’s consist :

- If the first unit is an ‘engine’, any ‘tenders’ following that ‘engine’ are regarded as part of that ‘power unit’.
- If the first unit is a ‘tender’, any further ‘tenders’ and the first ‘engine’ encountered after those ‘tenders’ are all regarded as part of that ‘power unit’. Any further ‘tenders’ following after the first ‘engine’ are not regarded as part of that ‘power unit’.

3.1.3. Notes on consist names for use in uncouple commands.

Even though a consist definition is required for each train, actually this information is often not used.
If a train is formed from another train, through a $forms or $detach command, it disregards its own consist definition, but instead ‘inherits’ the consist from the train from which it is formed. Each unit in that train or part of train therefore also inherits the original consist information.
Each unit has a variable “original consist”, this is set to the consist of which the unit is part when this unit is placed in the simulation. The unit keeps this “original consist” name during its full lifetime in the simulation, regardless of the train of which it forms a part.
It is this ‘original consist’ name which is used to determine which units must be detached if a consist name is used in an uncouple command.
Extra care should be taken in case units have (re)entered the simulation through a pool, in particular if the pool can hold units originating from different consists. A unit will keep its “original consist” information even if it has been stored in a pool, so the consist information for a unit exiting from a pool can not be known.

3.2. Detach command.

The $detach command defines that units are to be detached from that specific train. The detached units will form a new train as defined in the command. The train from which the units are detached will continue as defined.
If the $detach command is combined with a $forms command, the units will be detached first, then the train will form into the new train.

Command fields.
The $detach command can be defined in the following timetable fields :

- #note field : the units will be detached as the train is started.
- any station field : the units will be detached when the train stops at the relevant station. The detach will take place immediately as the train stops.
- #dispose field : the units will be detached when the train has reached its end position. The $detach command in the #dispose field **must** be preceded by either a **$static** or **$forms** command.

Command syntax.
$detach <unit details> <train details>

<unit details> can be any of the following :

- **/power** : all power will be detached - see note below.
- **/leadingpower** : the first power unit at the front of the train is detached.
- **/allleadingpower** : all power units at the front of the train are detached.
- **/trailingpower** : the last (rearmost) unit at the rear of the train is detached.
- **/alltrailingpower** : all power units at the rear of the train are detached.
- **/nonpower** : all units which are not part of the 'power units' of the train are detached.
- **/units=<n>** : the number of units as indicated are detached.
  If n > 0, the units are detached from the front of the train.
  If n < 0, the units are detached from the rear of the train.
- **/consist=<name>** : the part of the train which contains the units with the original consist name as indicated is detached.
  For details on use of the consist name, see general note above.
  The relevant units must form either the front or rear portion of the train, otherwise no units will be detached.

Note on use of **/power** and **/nonpower** qualifiers :
the program will automatically derive whether the power units are at the front or the rear of the train, and will detach the relevant portion.
These commands must therefore not be used if the train has power at both ends.
Take care with trains which have driving-trailers (cab control cars), as these are defined as 'engines' and are thus seen as 'power'. The **/power** and **/nonpower** qualifiers must not be used for such trains, e.g. push-pull trains and Multiple Units.

<train details> can be any of the following :

- **/forms = <newtrain>** : the detached units will be formed into the train as defined. Syntax for this command is similar as for the general $forms command.
- /**static[=<newtrain>** : the detached units will be formed into a static train.
  A name can be defined for this new train but it is not compulsory.

If the $detach command is combined with a $forms or $static command, the general $forms or $static command applies to the portion of the train which continues, the $forms or $static qualifier in the $detach command applies to the detached portion.

Examples.
$detach as part of a station stop command, where the train drops 3 units at the rear as new train :
$detach /units= -3 /forms=newtrain

$detach defined in #dispose command : all non-power units are detached as static train, the power units are formed into a new train :
$forms=newtrain $detach /nonpower /static

$detach defined in #dispose command : the train forms a static train, but all leading power is detached as new train :
$static $detach /allleadingpower /forms=newtrain

Note that these last two examples are two different ways to get to the same situation. Which definition is the best to use can depend on the situation and the composition of the train, but in general it makes no difference.

Detach command for player train.

When a $detach command is set for the train which is selected as player train, one of the three following situations will occur :

- Only the portion which continues contains one or more driveable engines.
  The player will remain with the present train.
- Only the detached portion contains one or more driveable engines.
  The player will switch to the detached portion, and will continue with the new train.
- Both portions contain one or more driveable engines.
  A window will pop up, informing the player that the train will split, in which portion the engine is located which the player is now driving, and what train the other portion will form into.
  The player can switch cabs and engines using Control-E, and the window will update automatically depending on the selected engine.
  When the player is in control of an engine in the portion with which the player wants to continue, the ‘confirm’ button in the window can be pressed and the detach will take place.

3.3. Attach Command.

The $attach command defines that a train is to attach to another train.
The attaching train will become part of the train to which it attaches, and ceases to exist.
It is not possible to attach to a static train which is not defined to start at a later time; in that situation the **$pickup** command should be used.

Command fields.
The $attach command can be defined in the following timetable fields :

- any station field : the train will attach to the defined train at that station.
  As the train ceases to exist, any definitions for that train beyond that station are ignored. Any dispose information is also ignored.

- #dispose field : the attach will take place at the end of the train's path.
  The $attach command may not be used in combination with any other dispose command.

Command Syntax.
$attach = <train> [/firstin] [/setback]
<train> : other train to which this train is to attach.
/firstin : this train arrives into the station as first train (see note below).
This qualifier can only be used for an $attach command in a station.

/backup : the train has to back up to perform the attach, i.e. the other train arrives behind this train. This qualifier can only be used in combination with the **/firstin** qualifier.

Notes.
When defining an **$attach** command in the **#dispose** field, it must be ensured that the other train is at the location at the end of this train's path. The attach can only take place at that location, even if the other train is in this train's path at an earlier location. This is to avoid accidental attachment in the wrong location. For instance, two trains follow each other some distance before reaching the location where the trains are to attach. If the first train is stopped, e.g. at an earlier station or at a signal, this would allow the second train to attach at that location, which is not what is intended.

If both trains arrive at the attach location within a fairly short time, it is advisable to set a $wait command for the second train so as to ensure the trains arrive in the correct sequence.
If the attach takes place at a station, the **$keepclear** command (see below) can be used to specify the stop location for the first train to arrive to ensure both trains will properly stop in the platform.

If the attach command is set for a station, the attach will immediately take place on arrival of the attaching (second) train.
If **/firstin** qualifier is set, the second train will stop the normal distance from the first train, and the first train will move toward the second train in order to attach as soon as the second train is at a standstill.

A train to which another train is to attach, will not depart from a station or from the location where the attach is to take place until that attach has actually taken place - in other words, in case the attaching portion is delayed, the combined train will also be delayed.

Attach command for player train.
If the player train is set to attach to another train, the player can just run up to that other train at appropriate speed and the attach will take place automatically. The player will switch to the train to which the player train is attached.

If **/firstin** qualifier is set, a note is set in the Next Station Window informing the player of the arrival of the second train, and an indication that the attach can be made.

If the player train is waiting for another train to attach, a note is issued when the attach is made. No action by the player is required. If the attach is to take place at a station, the Next Station Window will also display information when waiting for the attach to take place.

When the player train is the first to arrive for an attach in a station, please take note of the stop location information in the Next Station Window so as to ensure the train stops in the correct position along the platform.

3.4. PickUp Command.

The $pickup command is used to define a train to pick up a static consist. The train which is picking up will continue as defined, the consists which is picked up will become part of that train and will cease to exist as separate train.
Because the pickup consist no longer forms an actual train, $pickup can only be used to pick up static consists which have no more tasks within the timetable, as otherwise it would violate the consistency of the timetable. Such consists can be created by setting $static as train name, by using the $static command in the #start field, by using the $static command in the #dispose field or by using the /static qualifier in a $detach command. Note that only consists created using the $static command in the #start field can have a defined name, in all other cases the train in nameless.

Command fields.
The $pickup command can be defined in the following timetable fields :

- any station field : the train will pick up the defined train at that station.
- #dispose field : the pick up takes place at the end of the train's path.
  The $pickup command must be preceded by the $forms command.

Command Syntax.
$pickup = [<train> | /static]

<train> : defines the name of the train which is to be pick up, this can only be used in case of named static trains.
/static : defines that any consist at the relevant location will be picked up.

Pickup command for player train.
When a player train is to perform a pickup, the player only needs to run up to the waiting consist at an appropriate speed and attach to that consist. No further actions are required.
If the player train has formed into a static train (through a $forms or a $detach command), the player needs to wait for the other train to attach. As the other train attaches to the player train, the player will be switched to that other train. No further actions are required.

3.5. Comparison between $attach and $pickup commands.

Basically, both the $attach and $pickup commands perform the same action. In the case of $attach, the train performing the attach ceases to exist, while in the case of $pickup, the train which is attached to ceases to exist.
The commands are equal and there is no preference for using either one or the other.
As an example, take a train which terminates and forms into a new working (named newtrain). The incoming power is detached and runs to shed (as powerin), and later other power (named powerout) is attached to take out the new train.

This sequence can be defined using $attach with the following commands :

- Incoming train : #dispose command :
  $forms = newtrain $detach /power /forms=powerin
- Outbound power : #dispose command :
  $attach = newtrain

The sequence can also be defined using $pickup with the following commands :

- Incoming train : #dispose command :
  $forms = powerin $ detach /nonpower /static
- Outbound power : #dispose command :
  $forms = newtrain $pickup /static

These are not even the only options, other variations are also possible. In all, it makes no difference to the final outcome. What variation to use can depend on the location or situation but also on personal preference.

3.6. Transfer command.

A $transfer command defines the transfer of units between two trains. Both trains will continue to exist and proceed as defined.
A $transfer command is defined for one of the trains, this train is to arrive at the location as last train and will run up to the first train to perform the transfer.
In the details below, the train for which the transfer is defined is referred to as the *active train*, the first train is referred to as the *passive train*.
A transfer can take place in both direction, i.e. the *active train* can take units from the *passive train*, or it can give units to this train.

Command fields.

The $transfer command can be defined in the following timetable fields :

- any station field : the transfer will take place at that station.
- #dispose field : the transfer takes place at the end of the train's path.
  The $transfer command must be preceded by the $forms command.

Command Syntax.

$transfer = <train> <transfer details> <unit details>

<train> : name of the *passive train*, or **/static** if the transfer is to be made with a static train.

<transfer details> : defines what transfer is to take place. The units as defined in the <unit details> will be transferred as defined in the <transfer details>.
Options :
/give : units as defined are transferred from *active train* to the *passive train*.
/take : units as defined are transferred from the *passive train* to the *active train*.
/keep : units as defined will remain on the *active train*, all other units on the *active train* are transferred to the *passive train*.
/leave : units as defined will remain on the passive train, all other units on the *passive train* are transferred to the *active train.*

<unit details> : defines the units which must be transferred or kept as detailed.
Options :
/onepower : a single power unit is transferred or kept.
/allpower : all power units are transferred or kept.
/nonpower : all but the power units are transferred or kept.
/units = <n> : the number of units as defined are transferred or kept. Note that for transfer, <n> must always be > 0.
/consist = <name> : units which "original consist" value matches the defined consist name will be transferred or kept.

Note that these units must be at the relevant end of the train, otherwise the transfer can not be performed.

Notes.
For all unit details except the **/units** qualifier, if the first unit at the relevant end of the train does not match the required unit details, no units are transferred (e.g., /nonpower is defined but the first unit is an engine).
If the **/units** qualifier is set but <n> exceeds the length of the relevant train, all but one unit is transferred or kept as appropriate.

The *passive train* will not be allowed to depart from a station or the location where the transfer is to take place until that transfer has actually occurred.

Transfer command for player train.
If the player train is the *active train*, the train must be run up to the *passive train* at an appropriate speed. The transfer will take place automatically, no specific actions by the player are required. If the engine the player is driving is among the units to be transferred, the player will switch trains.
If the player train is the *passive train*, the player must wait for the transfer to take place before the train can continue. In this situation, as above, if the engine the player is driving is among the units to be transferred, the player will switch trains.

4. Start-up delays.

When a train is stopped, e.g. for a signal at danger, it takes some time for the train to restart when it is allowed to do so. This start-up delay has now been introduced for AI trains in timetable mode.
Different values are set for the various situations which can occur.
Each value consists of a fixed part and a random part. The actual delay which will be applied is a value in the range between [fixed_part] and [fixed_part + random_part].
The values are preset with default values as shown below. The default values can be overruled using the relevant parameters in the timetable file.
Restart after reversal is a special situation. An additional delay is set, based on the length of the train. This reflects the time required by the driver to walk to the other end of the train. The value of this parameter is multiplied by the length of the train to obtain the actual delay.

Default values.
Table below shows the various situations and the preset default values.
All values are in seconds except when indicated.

Table 1: Default values for start-up delays

Situation	Fixed Value	Random Value
Start of new train (“INI” phase)	0	10
Restart in “FOLLOW” mode when train ahead has restarted	15	10
Restart after stop for pathing reasons (e.g. signal, reversal)	1	10

Table 1: Default values for start-up delays

Situation	Fixed Value	Random Value
Restart after station stop	0	15
Restart after attach action	30	30
Restart after detach action	5	20
Additional restart after reversal (values in Sec / meter)	0.5	n.a.

Parameters.
The default values can be overruled by setting parameters for a sepcific train.
The table below shows the relevant parameters.
The parameters must be set in a row which is defined as **#restartdelay**.
Values are defined in seconds, except for additional restart after reversal which is set in sec\meter.

Table 2: Parameters for start-up delays

Situation	Parameter
Start of new train (“INI” phase)	$new /fix=<n> /var=<m>
Restart in “FOLLOW” mode when train ahead has restarted	$follow /fix=<n> /var=<m>
Restart after stop for pathing reasons (e.g. signal, reversal)	$path /fix=<n> /var=<m>
Restart after station stop	$station /fix=<n> /var=<m>
Restart after attach action	$attach /fix=<n> /var=<m>
Restart after detach action	$detach /fix=<n> /var=<m>
Additional restart after reversal (values in Sec / meter)	$reverse /additional = <n>

Future options.
At present, the delays are applied by default. It is intended that an overall user option will be introduced which can be used to switch off all delays.
Possible future options are that values can be set for one of the situations which will be applied to all trains or perhaps to certain trains only. Such values could perhaps be added to route files (.trk files) or consist files (.con files).

5. Speed control settings

In certain situations, the speed of a train may be restricted to a much lower value than allowed by track speed or consist speed. An example is, for instance, when a train is propelled out of a siding into a platform.

Furthermore, OR uses some specific speed values for AI trains in certain situations, e.g. minimum speed on approach to a signal (“creep speed”), speed when attaching trains etc. Also, there are situations where trains are normally running at a fixed speed somewhat below the allowed maximum, and only running at maximum allowed speed when the train is running late. This is introduced in OR as ‘cruise speed’.
It is possible to specify these values for indivual trains. Note that such restrictions are not passed on if a train forms into another train.

Speed fields.
A row can be set as speed definition using one of the following row identifications :
#speed : values are speed settings defined in m/s.
#speedkph : values are speed settings defined in km/hour.
#speedmph : values are speed settings defined in miles/hour.

Speed settings.
The table below shows the available speed settings, and default values if applicable.

Table 3: Speed settings

Command	Description	Default Value
#max	Overall maximum speed	-
#cruise	Normal cruise speed. Only valid in combination with #maxdelay command.	-
#maxdelay	Maximum delay (in minutes) for cruise speed.	-
#creep	Creep speed.	2.5 m/s
#attach	Speed when coupling to other train.	0.4 m/s

6. Pool concept.

The timetable mode offers the ability to keep trains ‘alive’ until they are needed again later. These trains, which may be light engines, multiple units or indeed full trainsets, are stored in sidings until they are required again for the next service. In busy areas, like yards, terminal stations and such, the number of units which are stored this way can be quite large.
At present, it is up to the builder of the timetable to ensure such units are send to and retreived from the storage in the correct sequence, which means a full diagram has to be worked out of when train are due to enter the storage area and when they are required again. This can be quite complicated.

The pool concept will take care of the storage requirements. When a train is send to a pool, the pool logic will work out where to store this train. On request of a train the pool will work out which train is available, and will release this train from the pool.
All trains in a pool are supposed to be the same, that is they are all interchangeable. They need not be the same trains as such, but be equivelant. It is not possible to set additional requirements when requesting a train from a pool.
So, for instance, if storage is required for both diesel and electric engines, this will require two pools, each with their own storage area.

Definition.
A pool is identified by a unique name.
Each pool contains one or more storage areas, defined as a path. Only dead-end storage areas are supported (ie there is only one entry/exit point per storage path).
Each storage area is linked to one or more access paths, which define the path from an exit/entry point to or from the storage areas. All storeage paths must have an access path to all defined exit/entry points. There can be multiple exit/entry points, but each storage area needs to have access to all these defined points.
Path of trains to or from the storage area must only be defined to or from an entry/exit point.
As pools are part of a timetable definition, the definition file needs to be stored in the same directory as the timetable files (subdirectory OpenRails in the route's Activity directory), and as timetable files it needs to have unicode .txt format, with file extention .pool-or.
There can be multiple pool definition files, and each file can contain multiple pool definitions.

Train commands.
A train can be send to a pool by the command $pool=<name>, set in the #dispose line.
A train can be retreived from a pool by the command $pool=<name>, set in the #start line, preceded by the time at which the train is required.
A train can be defined as starting in a pool by the command $create /pool=<name>, set in the #start line, without time. The train will be created at the start of the timetable.

Testing and evaluation options.
When there is a pool 'underflow', no engine is available and therefor no engine is send out on the request for an engine release. A user option is envisaged, "force create train on underflow", which will forcibly create an engine when this situation occurs. This can be used during testing of a timetable, when it is not yet quite clear how many engines are required.
Further options may be defined for pool evaluation. How this will be done exactly is still being considered. One possibility is to set an option to print all pool details to a specific file, these details will contain timings on when engines enter and leave the pool, total no. of engines in the pool after that action, total available space, and underflow and overflow situations.
Another possibility is to add this information as a new page to the F5 hud info.

7. Additional commands.

This chapter lists additional commands which have not been described in the chapters above.

#note field.

$doo

sets train as "driver only operation".
This only affects the playing of the 'departure sound' when the train is ready to depart from a station. When $doo is set, the sound is not played.

$forcereversal

will set the reversal point of a train to the 'diverging point' and will disregard any signals in the train's path.
Normally, if there is a signal facing the return direction of the train between the diverging point and the defined reversal position, the actual reversing position is placed at this signal.
If $forcereversal is set, such a signal is ignored and the actual reversing position is

placed at the diverging point. This can be usefull in areas with limited signalling, e.g. when shunting within a yard, the train does not need to go all the way out to the yard entry signal.

station stops fields.

As command for stations only :

$stoptime = <n>

Sets the default station stop time to <n> seconds.
This command overrides the default stop time as defined in the .tdb file.
The default stop time is applied to all trains.

As command for stations and train stops :

$restrictPlatformToSignal

Can be used in situations where a signal is placed inside the platform markers.
Setting this command will restrict the platform to the location of the signal.
This ensures that the station stop will prevail over the signal stop in case the signal is at danger.

$extendPlatformToSignal

Can be used in situations where the length of section available as platform is not fully covered by the signal markers, e.g. because there is a junction along the platform.
Setting this command extends the platform to the first signal following behind the platform marker in the train's route direction. However, the platform will never extend beyond the actual train's route.
This command is particularly usefull in loops where the actual platform length is restricted due to access tracks to sidings etc., it will allow the use of the full available track length between signals as platform.

As commands for train stops only :

$closeupsignal

This will set a shorter stop distance to the end signal of a platform instead of the default value. The shorter distance will only be set if required to position the train along the platform as required, e.g. due to train length or because of a *$keepclear /rear* command.

$keepclear

This command defines a specific location for a train to stop.
It can be used to keep enough platform length clear for other units to attach, or it can be used to ensure the train clears a specific switch or signal.
The command has the following parameters :
/rear = <n> : required platform length to be kept clear at rear of the train
/front = <n> : required platform length to be kept clear in front of the train
/force :
if used in combination with /rear :
allows the train to run beyond the end signal of the platform, see below for details.
if used in combination with /front :

will keep required distance clear even if end of train does not fit in platform.

Use of /force in combination with /rear.
If the train does not fit into the platform when leaving the required length of platform clear at the rear, setting /force will allow the train to run beyond the platform end and through any signals beyond the platform, provided ofcourse that the signals are cleared.
The train will, however, never run beyond the defined path.
If the station stop is the last stop of the train, the additional command $endstop must be set to ensure the stop is seen as the end of the route of the train.
If $endstop is not set and the route extends beyond the stop position of the train, the train will depart from the station and will terminate at the end of the path.

$endstop

To be used in combination with $keepclear /rear - see description above.

$forcewait

Will force a wait command on the first signal beyond a platform even if this is not the actual ‘end’ signal for that platform.
In such a situation, a normal $wait command would be set on a switch and not on a signal.

$noclaim

Will prevent the train from ‘claiming’ sections when held at a signal.
Usefull at busy locations where trains claiming routes when waiting for signals to clear could cause a lock-up.

#dispose field

Additional qualifier for $forms and $static commands :

/closeup

Will position the train close to end of track or other train.
/closeup does not affect the position of a train if the stop position is a signal or a switch.