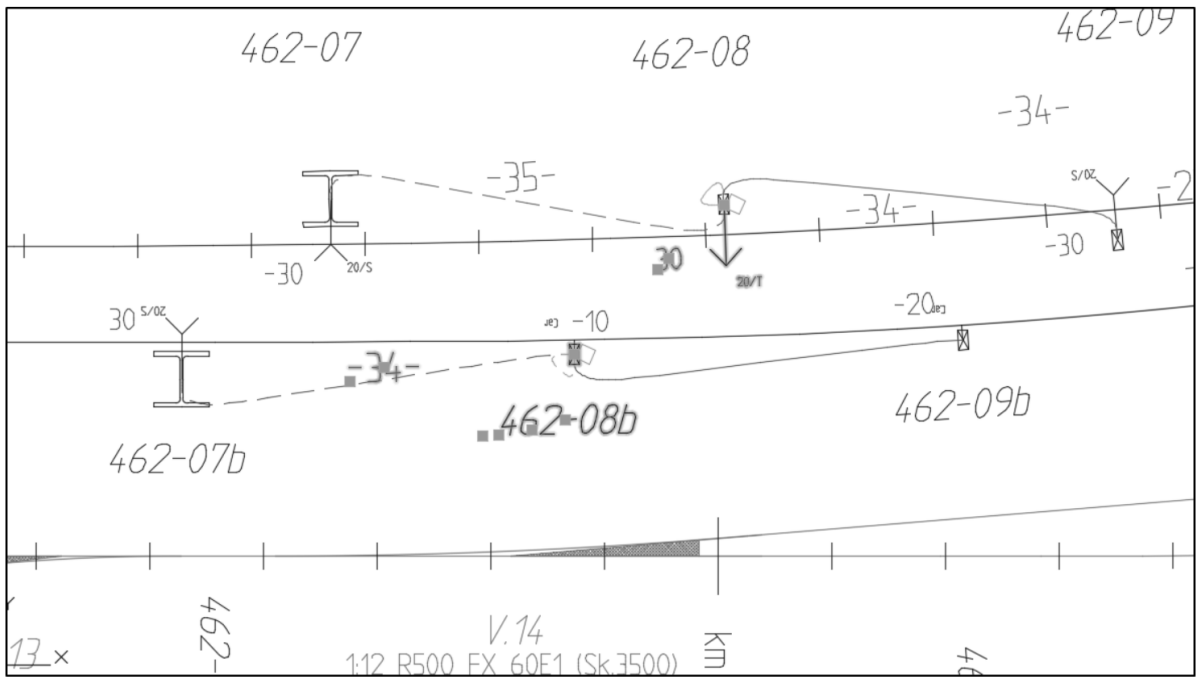


Introduction to RailCOMPLETE v2021.0



Contents

Introduction.....	5
Scope	5
Contact us.....	5
Download and installation	6
License levels.....	6
Beta testers	6
Loading the RailCOMPLETE ribbon.....	6
Note to non-English users	8
Using RailCOMPLETE	10
Commands.....	10
Patches	10
The Command Browser	10
Getting help.....	11
Example – creating railway tracks.....	12
Start a new RailCOMPLETE document	12
Import alignments from LandXML	13
Work with annotations.....	17
Copy annotation to drawing.....	18
Insert continuations, switches and crossings.....	18
Add fouling points	19
Work with alignments	21
Define reference alignment	21
Changing parent alignment.....	24
Point objects.....	27
Insert object	30
Positioning the signal	31
Attaching a point object to another point object – making a family	33
Copy point objects.....	35
Managing Objects.....	39
Autonumbering	39
Property names versus property displaynames.....	40

Tables	42
Create predefined table	42
Modify table	42
Export to 3D.....	45
Typical workflow – 2D vs 3D.....	45
3D preview tool	45
3D export example	46
Object properties	50
Building Information Modeling (BIM)	50
General data entry using the keyboard.....	50
Modifying multiple objects simultaneously	50
Data entry using the mouse scrollwheel.....	50
Lua programming	52
Lua example	52
Formula or Function?	53
Return values in formulas	54
Intrinsic RailCOMPLETE functions	55
Native Lua functions.....	56
What if a property name is the same as a reserved Lua keyword or identifier?	58
Custom Lua function declarations in the DNA	58
Lua function naming.....	58
Lua function naming conventions in RailCOMPLETE.....	59
Constants.....	61
Arguments	61
Programming style – Way Of Working (WOW!).....	62
Conventions for Model Check functions	62
Variables, functions and their scope	63
Refreshing objects or not	63
Copy formula	64
A simple Lua program.....	65
Scripts in Lua.....	69
RC-RunScript and RC-EditScript.....	69
Sandboxing	69



The Lua scripting API	69
Script commands reference manual	70

Introduction

Scope

This document is an introduction to let you understand and use the main features of RailCOMPLETE.

This document is meant for self-study, for use in student workshops or in the introductory RailCOMPLETE beginner's one-day course.

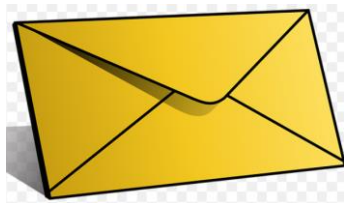
You will meet these concepts:

- Commands
- LandXML track design import
- Alignments and reference alignments
- Point objects
- The Manage Alignments tool
- The Manage Properties tool
- The Manage Objects tool
- Tables and the table editor
- 3D
- Lua and Lua naming conventions (yes, you will need Lua skills)
- Relations
- Cross-referenced models
- DNA and agents

The purpose of this document is to make new users become familiar with how to use the software. The RailCOMPLETE Reference Manual contains more detailed information about all features in the application.

We recommend both newcomers and rookies to attend our courses. Contact your local dealer or your local RailCOMPLETE web site for details.

Contact us



Your comments are very welcome! Please send your comments and suggestions to:

support@railcomplete.com

Getting started

Download and installation



RailCOMPLETE needs to be installed alongside AutoCAD®. On our download pages you will find details on how to install and start the software and how to extend your license.

<https://www.railcomplete.com/en/download>

On our download pages you will find the most recent additions and updates to our self-study tutorials. Many of these are bundled with RailCOMPLETE. Take a look from time to time to see if there are new or updated tutorial files there, or a more recent version of RailCOMPLETE. Your license code does not have to be keyed in again after installing a new RailCOMPLETE version.

You may use any version of the RailCOMPLETE software application. You will also need a suitable 'DNA' – Definition of Network Assets. If you can't see any DNA names when you click the RC-Start button, then you must locate a suitable one, download it and install it. We recommend starting with the fictitious 'Genericland Railways', abbreviated 'XX-GL', which contains a set of bundled tutorials.

License levels

Regular users (with a low license level) will see a white RailCOMPLETE icon in their top left corner. Development version of the software application display an orange icon with "DEV" written across it. Only users with a "DEVELOPER" license level will have access to all the commands under development. The higher your license level is, the more functionality you will have access to.



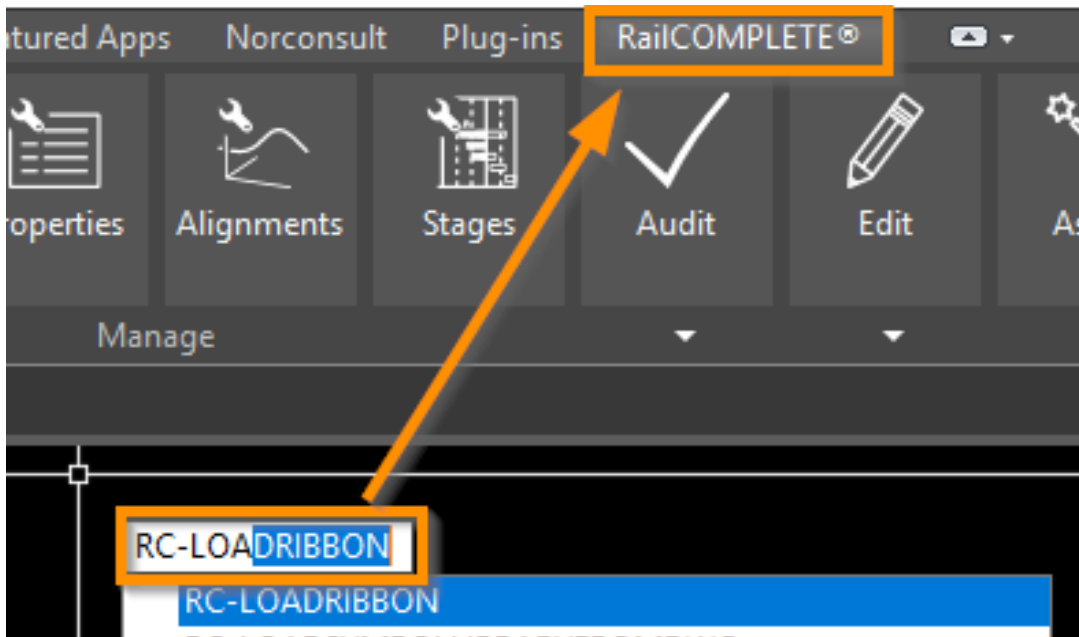
Beta testers

If you want to become a beta tester, please send us an email to support@railcomplete.com and we will consider upgrading your license level. The BETA versions will display an orange icon with "BETA" written across it.

Loading the RailCOMPLETE ribbon

RailCOMPLETE is loaded when AutoCAD is started.

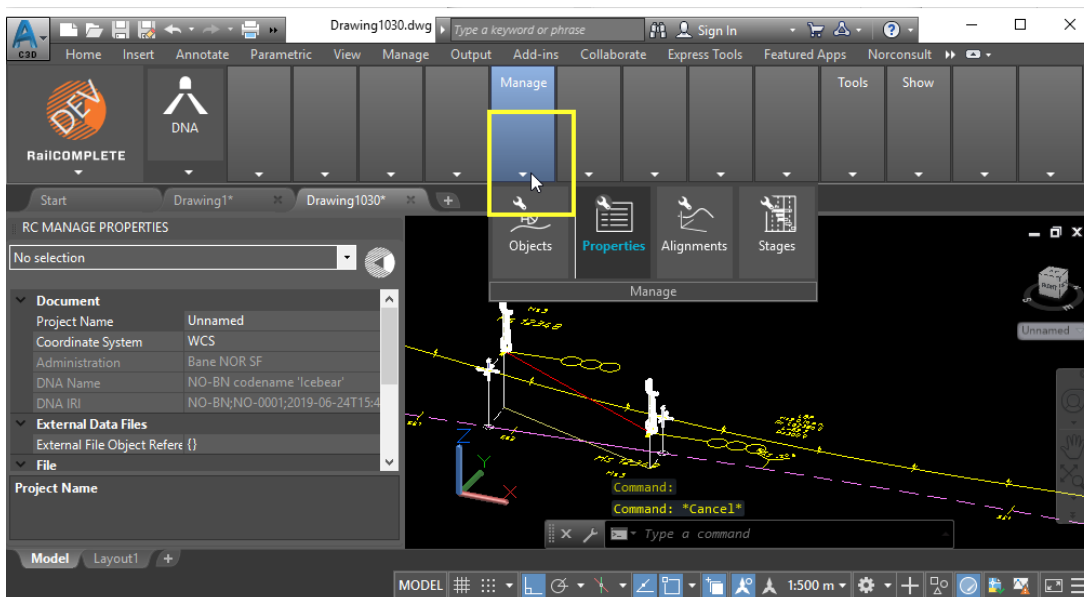
Depending on your AutoCAD profile setup, the RailCOMPLETE buttons and menus will become directly visible under the RailCOMPLETE menu caption. If the RailCOMPLETE ribbon is not visible (a top row on your screen, with command buttons) then type in the command name 'RC-LoadRibbon' and check that buttons and menus appear.



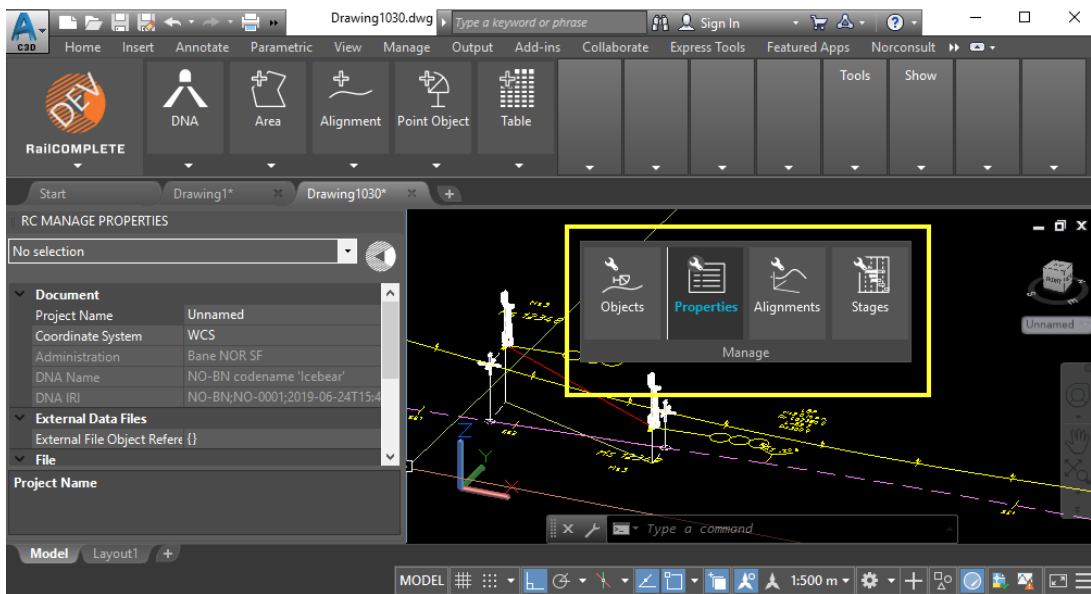
If you are using a computer with multiple screens then you must ensure that the magnification level (100%, 125% etc) is the same for all screens. Otherwise, you will experience garbled windows, weird docking of windows etc.

If you have disconnected your laptop and been to a meeting without exiting AutoCAD, then your RailCOMPLETE menus may appear garbled when you dock into your multi-screen workplace again. Run the command RC-ReorganizeRibbonToDefault to fix this problem.

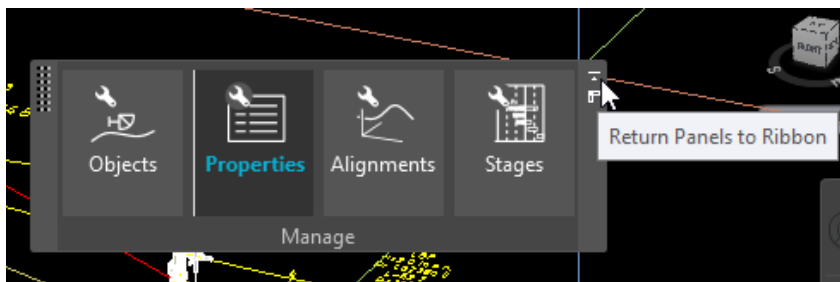
If your screen isn't big enough to show all the ribbon icons (the buttons are narrower and lose their icons) then you may undock entire button groups and place them somewhere on your screen. Hold down the Ctrl key while you drag and drop a button group.



After moving the 'Manage' button group using Ctrl + drag&drop:



Click the top right corner to return the undocked button group to the ribbon:



Note to non-English users

Command names in RailCOMPLETE are dependent on the AutoCAD language pack that you have installed. To use the English command name from a non-English language pack installation, precede the English command name with an underscore character, '_'

Your AutoCAD session has probably been started from a Windows shortcut of the type:

"C:\Program Files\Autodesk\AutoCAD 2021\acad.exe" /product ACAD /language "fr-FR", where "fr-FR" means "French language, France's version", or similar.

Native AutoCAD commands may have different names in your language pack, other than the COPY, COPYBASE, FIND etc that you see in our tutorial texts. To instruct AutoCAD to accept the native English command name, precede the native (English) command name by an underscore character, '_'. For instance: '_FIND' will start AutoCAD's native 'FIND' command even if you are using AutoCAD with the French language pack, where the command in French is called 'RECHERCHER'.

If a command needs an argument 'ON', and the French menu says 'Activer', then you can enter '_ON' to instruct AutoCAD to use the option's native name.

Furthermore, the English AutoCAD object selection prompt (command _SELECT) accepts many keyboard shortcuts such as A = (add) add to selection set, R = (remove) remove from selection set and AL = (all) all objects (and many more). These shortcuts are named differently in other language



packs. In French they are for instance A=ajouter, S=supprimer, TO=tout. Consult AutoCAD Help in your native language.

You may have seen a dash '-' that precedes command names – this is used to suppress the window dialogue of an AutoCAD command and instead offer the command-line interface. This is useful when commands are run from a from a Lisp program, or called using the runCommand() API call from a Lua script running under RailCOMPLETE.

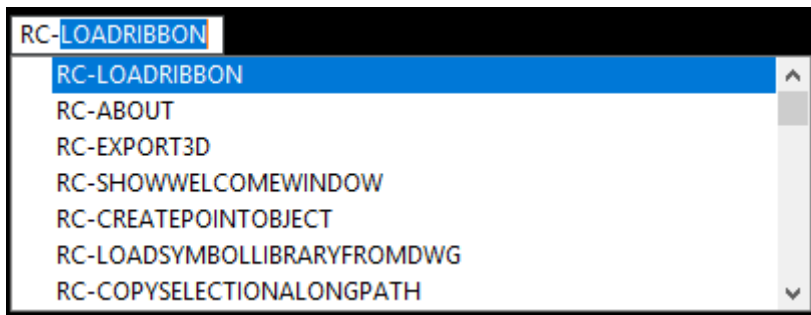
Using RailCOMPLETE

Commands

Most of the functionality in RailCOMPLETE is command based. Like AutoCAD®, commands are executed by typing in the command name inside the modelspace (the dark area where you enter data, as opposed to 'layouts' where you prepare paper or PDF drawings based on selected views into modelspace – consult AutoCAD tutorials to learn more about modelspace and layouts).

All RailCOMPLETE commands are prefixed with “RC-”.

Command names will generally be made up of a sequence of words starting with an action and ending with an object, as in 'RC-CreatePointObject'. There are a few exceptions to this rule: RC-About, RC-Help and RC-Settings.



RailCOMPLETE commands are universal in the sense that they have been developed with a level of generality allowing them to be used with any railway administration in the world. There are a few very specific commands, such as RC-DisplayBrakingCurveEbicab700 which is a supplier-dependent ATP braking curve algorithm.

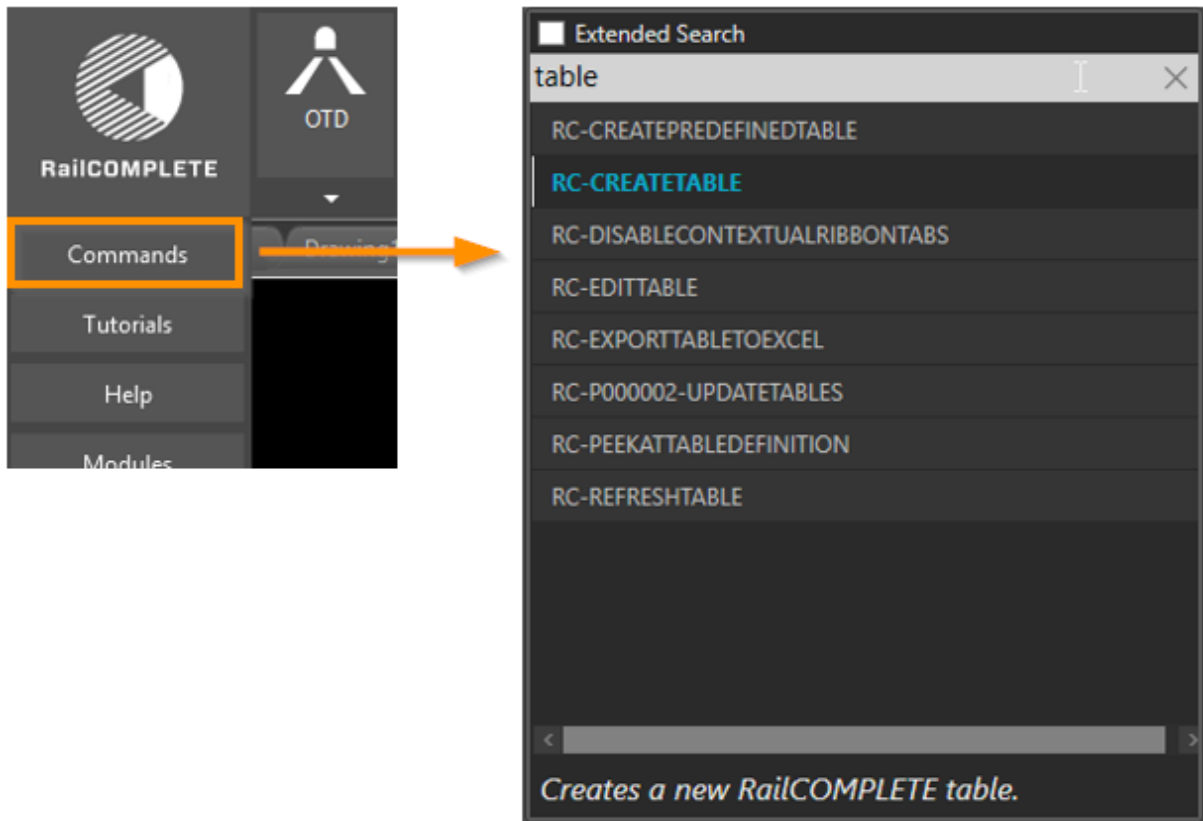
Concepts which are the same but use different parameterization across countries and railway administrations will be split such that the generic part is carried out by the RC command but the parameters driving the command's functionality are stated in each administration's DNA – Definition of Network Assets. See more about DNAs later in this document.

Patches

From time to time there will be a need for salvaging customer's old data files in ways that are not possible, or very cumbersome, using the standard file update mechanisms of RailCOMPLETE. In such cases you may ask for a specific 'Patching' command which does a very specific operation on your data as specified by you, or by us on your request. You will find them under the name 'RC-Pnnnnnn-xxxxx' where nnnnnn is a six-digit integer which increases from 000001, and xxxxx is an optional descriptive text. To be used with caution.

The Command Browser

By selecting “Commands” below the RailCOMPLETE icon, a searchable command list is opened. Commands can be executed by double-clicking or by pressing <enter>.



Getting help

All buttons in RailCOMPLETE have a brief and/or a more detailed description behind the tooltip. To see the short text, hover over the ribbon button or pull-down menu button. To see the more elaborated text, just hold your cursor until the longer text appears.

Many commands have more detailed information in the help system. The help system is launched by pressing F1, while hovering over the menu item.

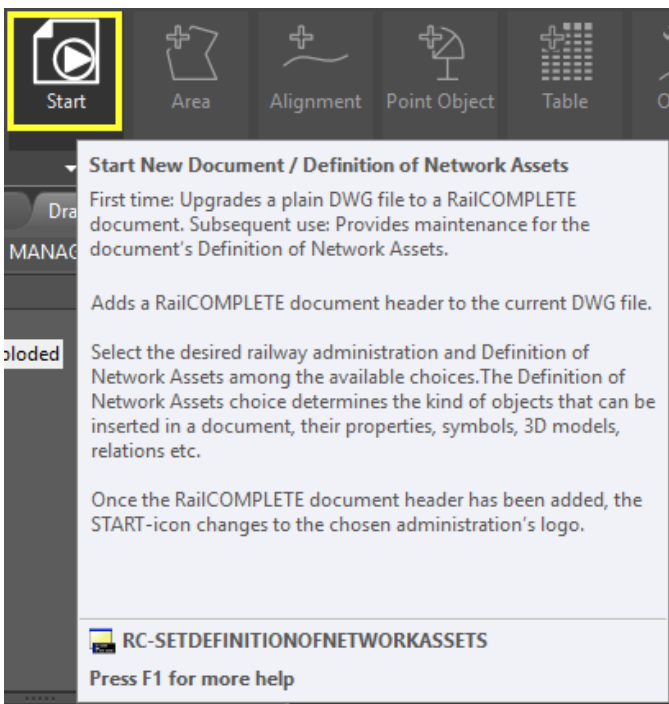
Example – creating railway tracks

This section will show the process of importing alignments from LandXML and how to use basic functionality in RailCOMPLETE.

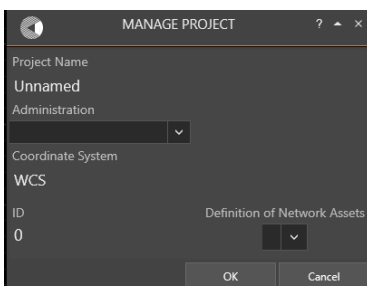
Start a new RailCOMPLETE document

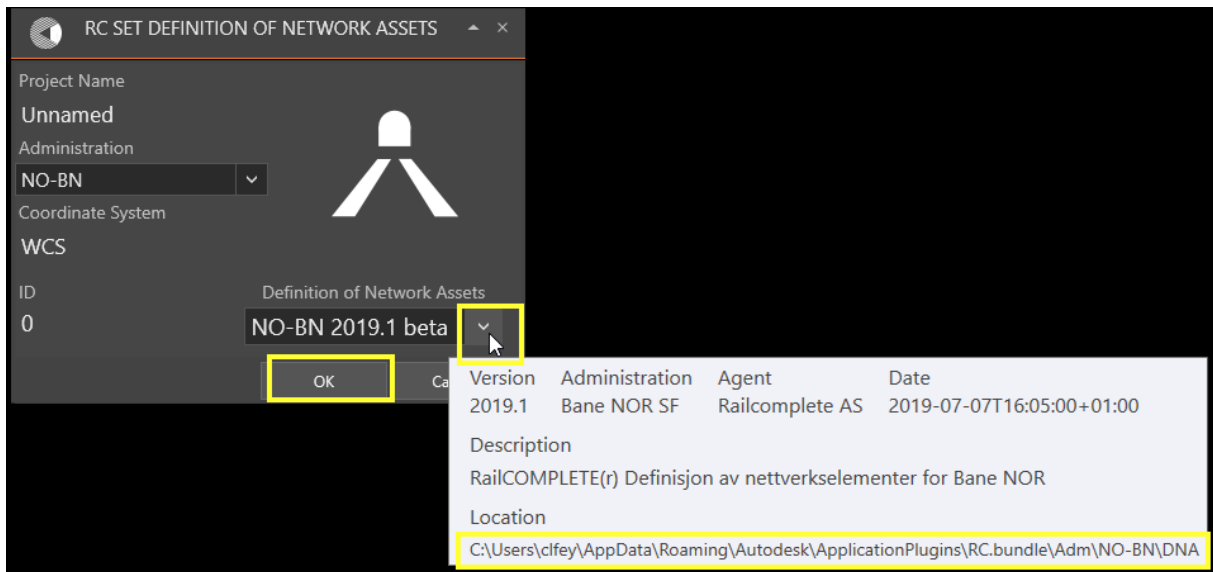
The document (your AutoCAD DWG drawing) must be converted to a RailCOMPLETE document to utilize the features within the RailCOMPLETE application.

To start a document, select the 'Start' button and select your preferred railway administration and DNA version¹. Hover over the little down-arrow to see details for each DNA candidate. In this tutorial, just use the shown default values and click OK.



¹ If you can't locate a suitable DNA when you click on the RC-Start button, then download and install one from our web pages. When no DNAs are installed on your computer then your start window will look like this:



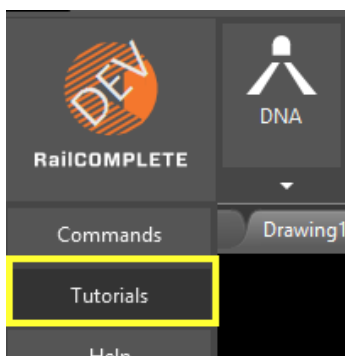


The RailCOMPLETE icon rotates while the selected DNA is injected into the drawing's 'kernel', as if it were an empty cell in a living body, now enabled with our selected DNA. When the rotation stops, RailCOMPLETE has complete knowledge about your administration's object types with their standard values and methods.

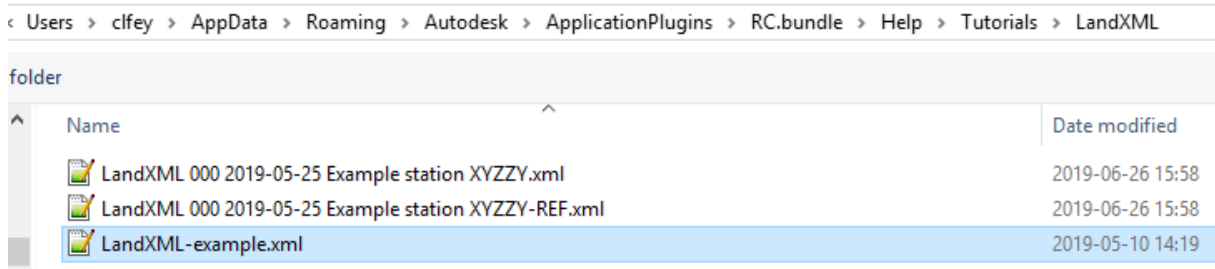
Each time you change between open documents (files) in AutoCAD, RailCOMPLETE will reload the DNA from the file's kernel. If you are an advanced user and you have made mutations to a file's DNA, then you must save it to the drawing's kernel before you close the document or move to another one, otherwise your DNA mutations will be lost.

Import alignments from LandXML

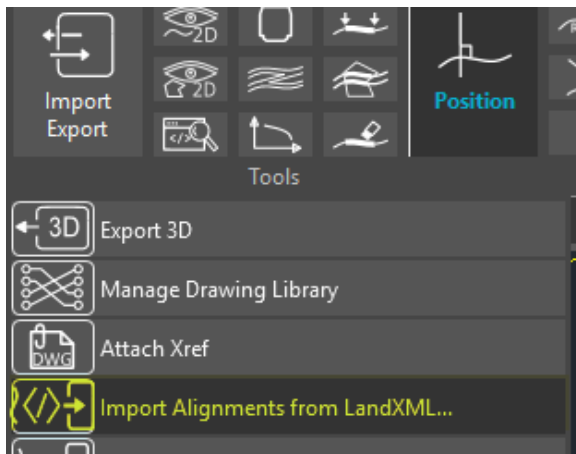
To locate your folders containing the bundled tutorial files, click the 'Tutorial' button below the RailCOMPLETE icon.



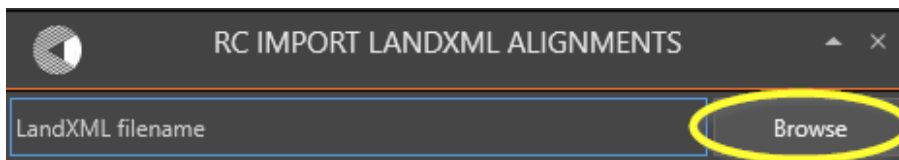
The file needed in this tutorial is stored in the 'LandXML' folder under the 'Tutorials' folder.



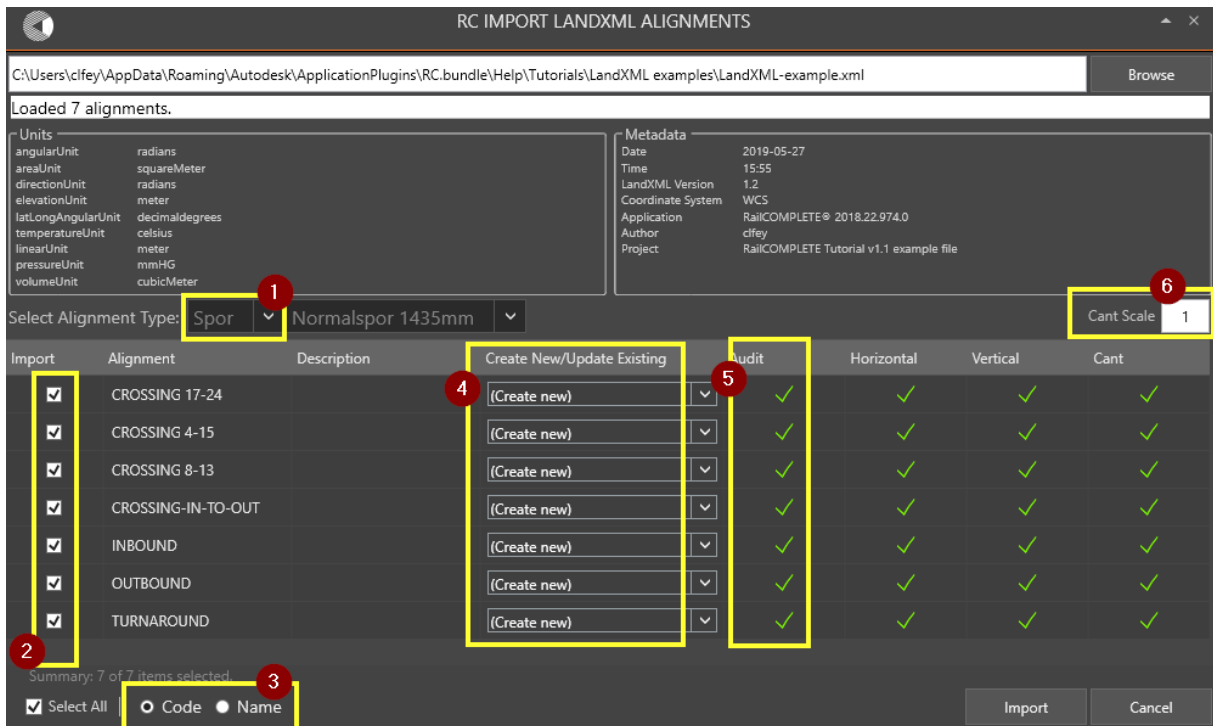
Start the LandXML import tool from the 'Import Alignments from LandXML' button below the Import/Export button.



The LandXML alignment import tool opens. Click 'Browse'.

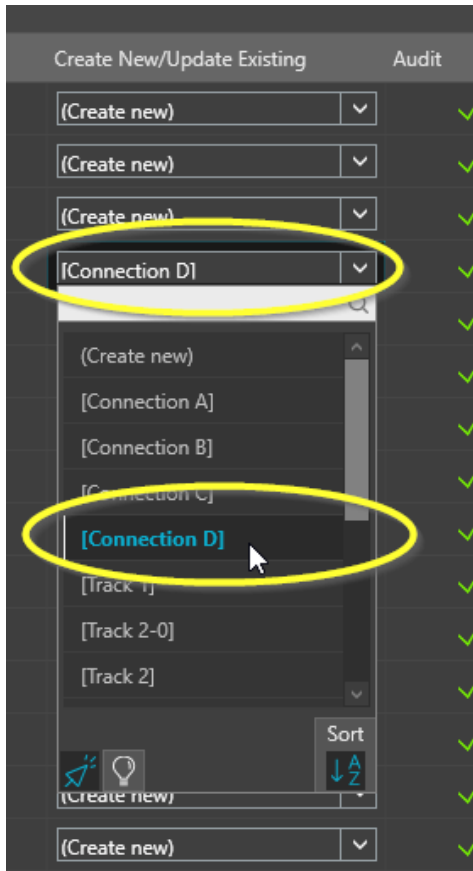


Locate the example file named " LandXML-example.xml" from the Tutorials\LandXML folder bundled with your installation. The LandXML file is scanned and analyzed, and the contents are displayed as shown below:



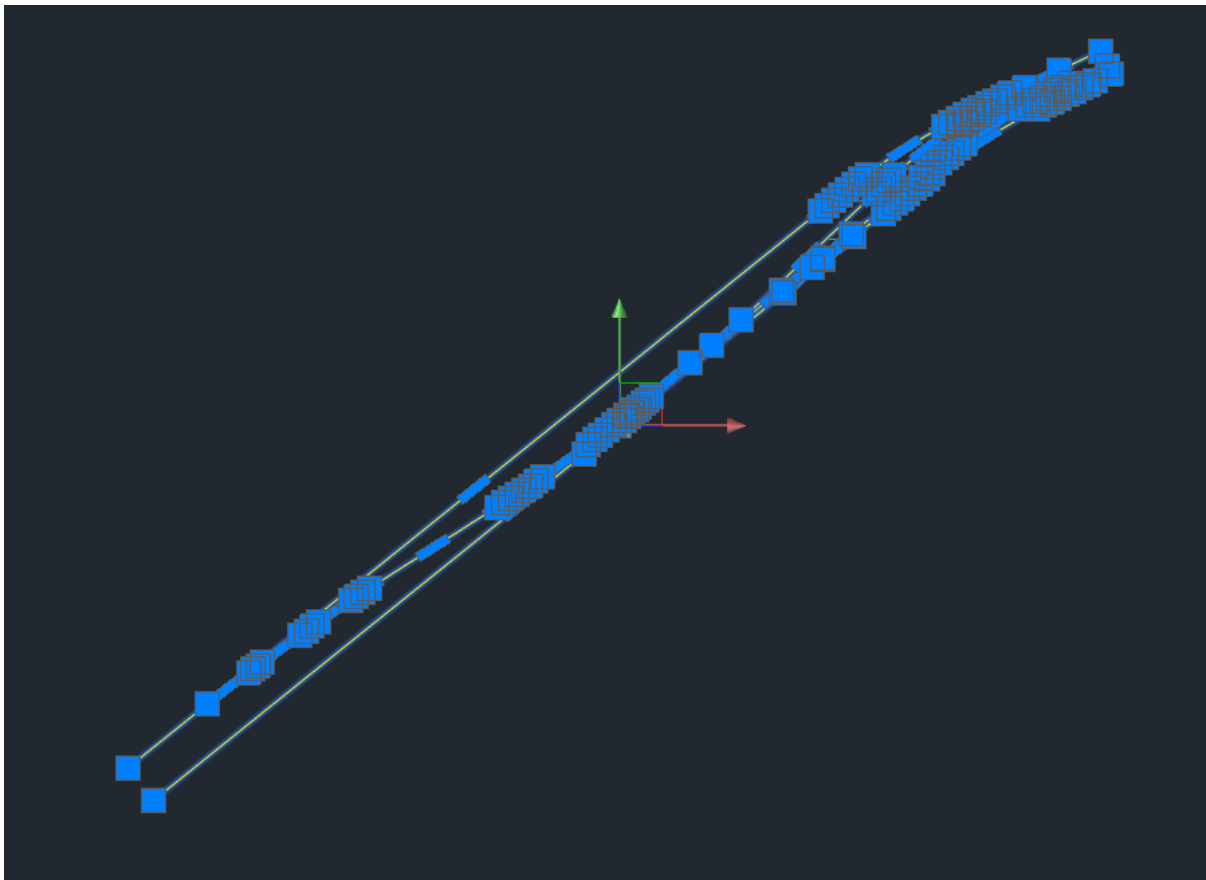
1. Make sure that you import the alignments as Railway tracks, select “Spor” (meaning “tracks” in Norwegian)
2. Select all alignments (default) or just the alignments that you want to import
3. The alignment names found in the LandXML file can be directed to either the RailCOMPLETE alignments’ ‘code’ or ‘name’ property, use the radio buttons Code/Name to select
4. “(Create new)” means that no alignment existed beforehand with the same ‘code’ property. This comparison is carried out only if you have selected the RC ‘code’ property as recipient for the LandXML alignment’s name. If an alignment already exists with the same ‘code’ property, then you will automatically be offered to update (overwrite) the existing alignment, keeping the same object id (GUID²) as before. You may all the same choose to create a new one – even with the same ‘code’, their IDs will be different. See the illustration below.

² GUID (UUID): globally (universally) unique ID. RailCOMPLETE objects use 256 bits presented as 32 hex characters in a 8-4-4-4-12 pattern, e.g. ‘827946bf-73f9-4c6c-87c7-542208d377bf’.

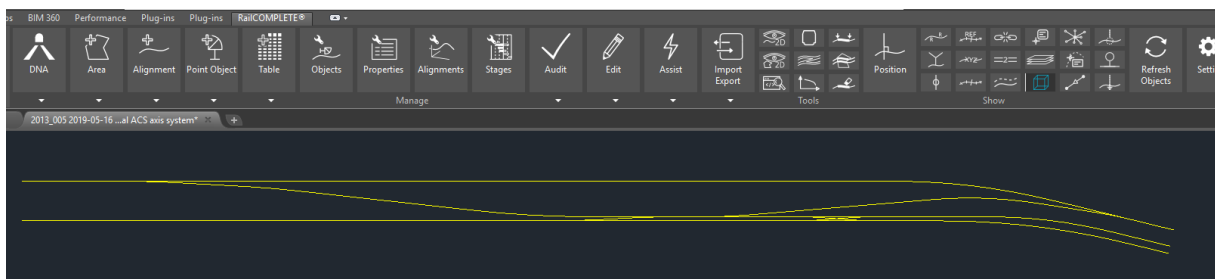


5. RailCOMPLETE uses only mandatory LandXML data when importing alignments. However, many derived values may be found in the LandXML file. If inconsistencies or errors are found, they will be flagged here. You may use the command RC-AuditAlignments later to locate and fix most common errors, for instance the presence of millimeter-short segments which give rise to loss of computing precision with ensuing lousy direction estimates.
6. Some alignment design packages erroneously export cant data in meter unit instead of millimeter. Set the cant scale to '1000' instead of '1' to fix this on import. Or just set it to '10' to exaggerate your track designs to better see in 3D visualizations where you have superelevations.

The result should be something like this:



Use the command RC-RotateUcsAndView to rotate the alignments to a better viewing perspective:



You can find this useful command in the context menu as 'Rotate UCS and View' or 'RailCOMPLETE View/Rotate UCS and View' (the context menu changes appearance depending on whether you currently have selected objects or not).

Work with annotations

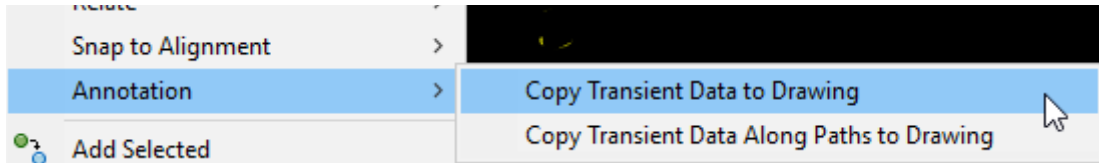
Many helpful annotation tools are placed in the "Show" part in the ribbon. Some act on alignments, some act on CAD layers, some act only on point objects. Select one or more alignments and try.



Toggle some of the annotation buttons to see what happens. The level of details shown will change during zoom. Some annotations are only visible above a certain zoom levels. Annotations will be suppressed if there are too many objects to show, this limit may be set using RC-Settings.

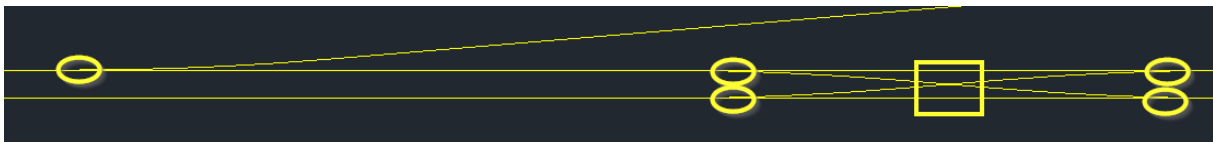
Copy annotation to drawing

The annotations are only transient graphics. To make the annotations permanent in the drawing with a fixed size, select “Copy Transient Data to Drawing”. Make sure the zoom-level is acceptable before making the transient annotation permanent.

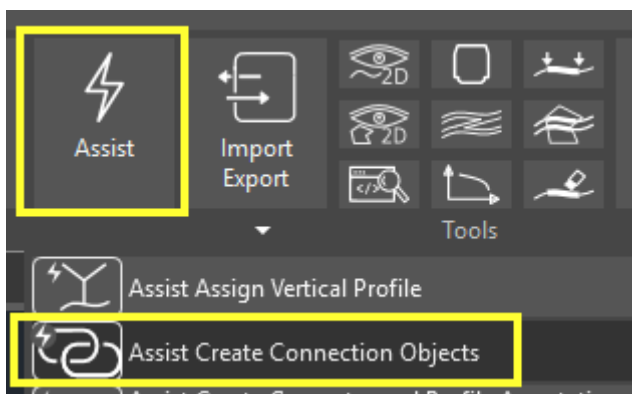


Insert continuations, switches and crossings

RailCOMPLETE will search for possible connections along the alignment geometry (dealing with railway tracks) and insert track continuations, switches (turnouts, points) and crossings automatically. A ‘continuation’ occurs whenever one alignment apparently stops but another one starts (or ends) very close. A ‘switch’ (see the circles below) occurs whenever an alignment appears to ‘break off’ from another. A ‘crossing’ (see the square below) represents the situation where two alignments cross, and none of them ends close to the crossing point. RailCOMPLETE stores standard geometries for each modeled administration and uses this information to judge what type of switch geometry is found.

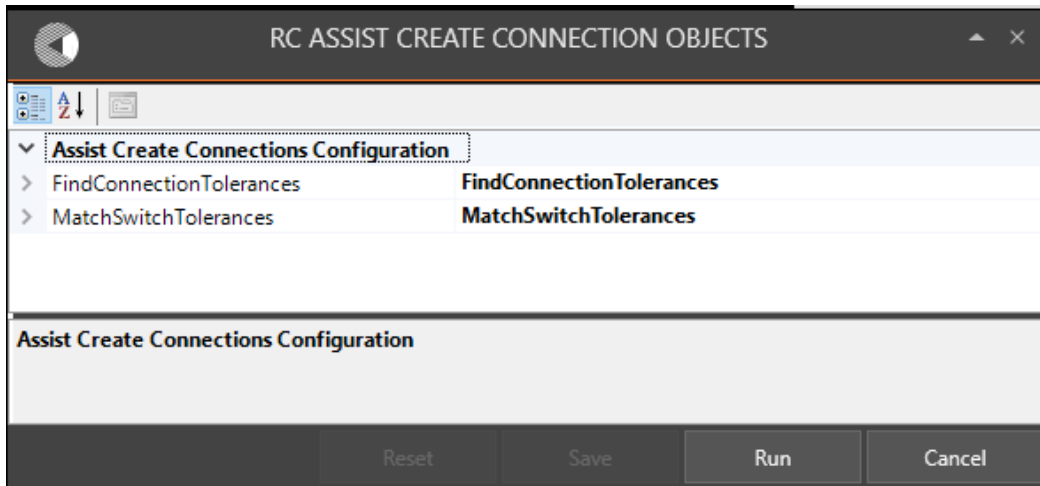


Start the RC-AssistCreateConnectionObjects and answer select “Insert All Switches”.

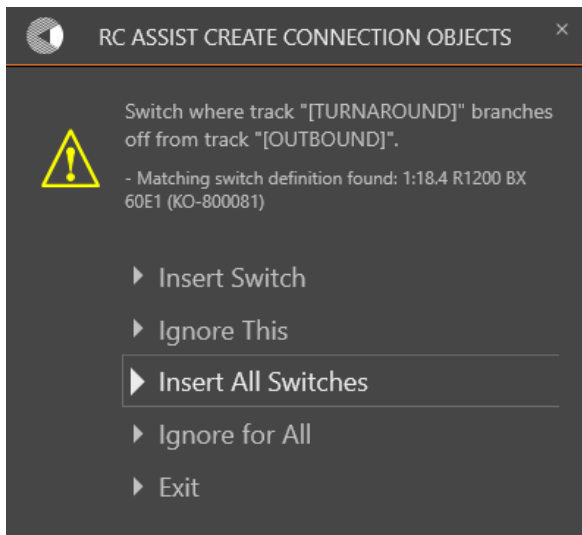


You will be shown a dialog where you can adjust tolerances for the geometry pattern matching algorithm. Make and save your selections and then press ‘Run’ to continue. You may at any time revert to default settings for the current administration.

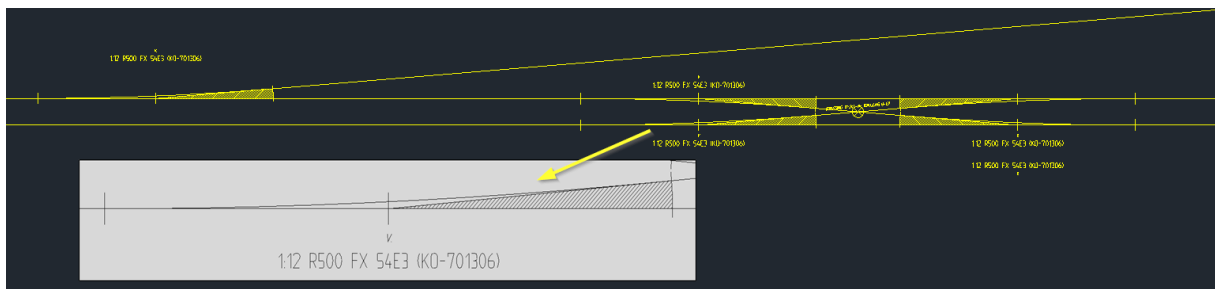
Note that any alignment will be treated as a candidate. If your model contains other alignment types than railway tracks, just freeze or turn off their respective CAD layers before you run the connect-objects assistant.



Accept 'Insert All Switches' when prompted.



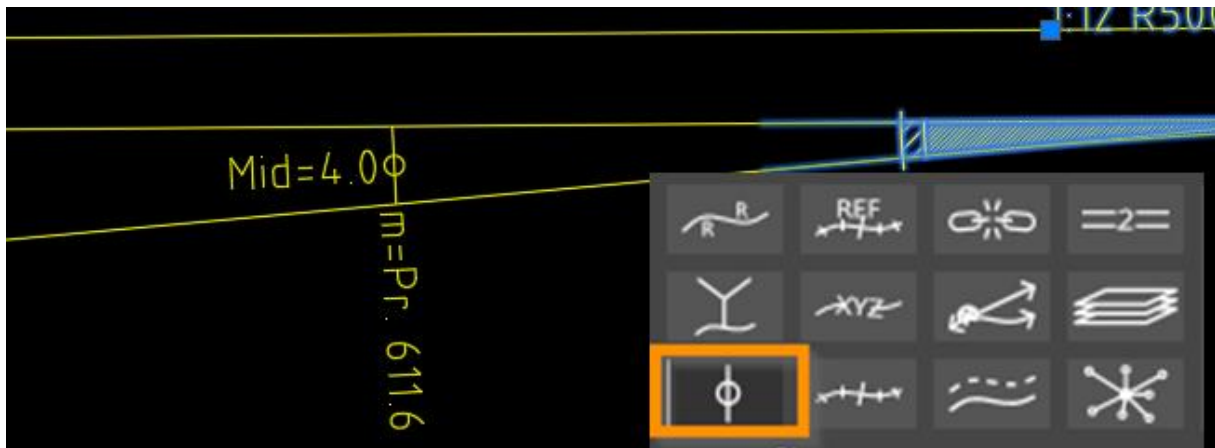
The connection objects are now created, and the drawing should look something like this.



Add fouling points

Now that the drawing contains switches, it is possible to add fouling points (the danger point, after which two trains would be too close to each other in a switch).

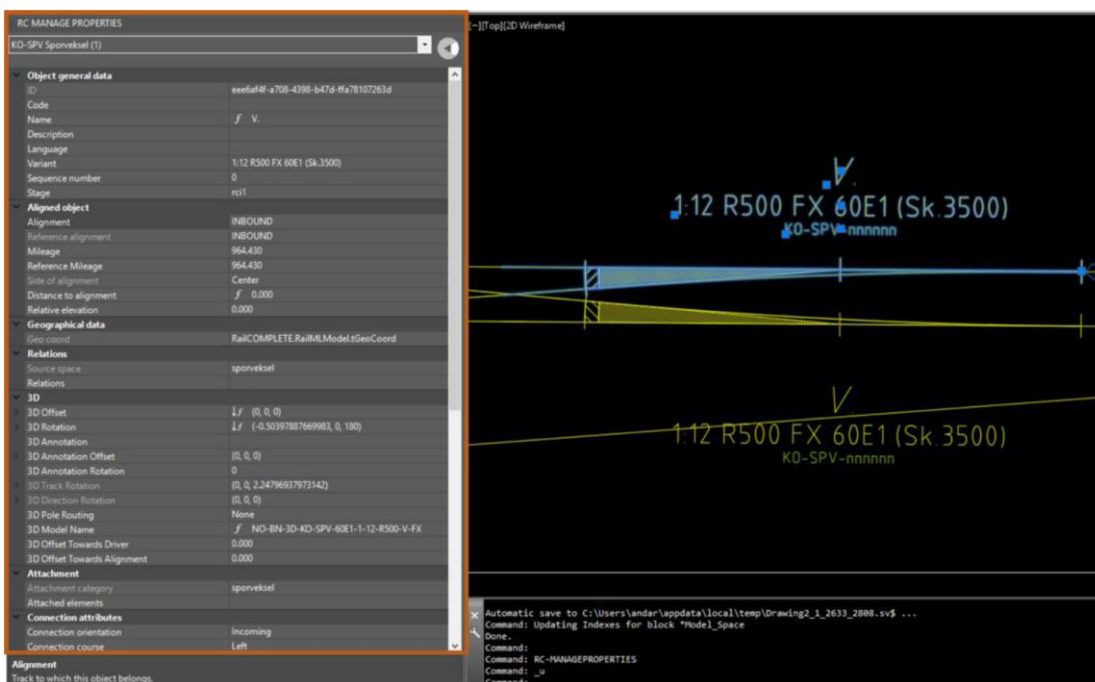
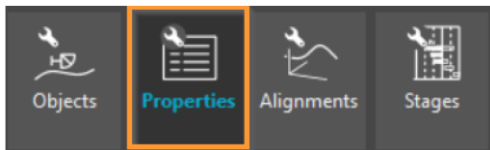
Toggle the fouling-point tool button and select one of the switches to see the fouling point annotation.



Make the fouling point illustration permanent in the drawing by zooming to the desired detail level, right click and select “Copy Annotations to Drawing”.

Select one of the switches and have a look at the RailCOMPLETE properties for the switch.

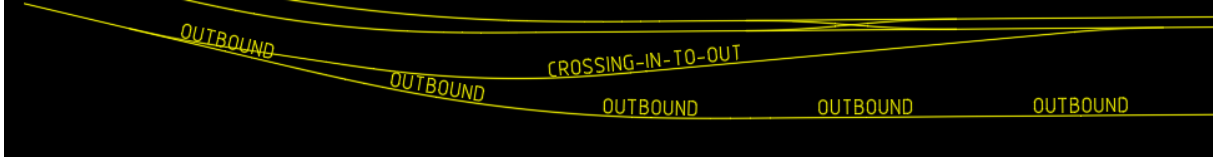
The RailCOMPLETE properties window is opened by selecting ‘Properties’ in the ribbon, or by typing RC-ManageProperties in the model space command window in AutoCAD, or by double-clicking a point object (the double-click behavior may be adjusted using RC-Settings), or by using the RC-CommandBrowser tool to locate the right commands – or by right-clicking and selecting RailCOMPLETE Manage -> Properties from the context menu.



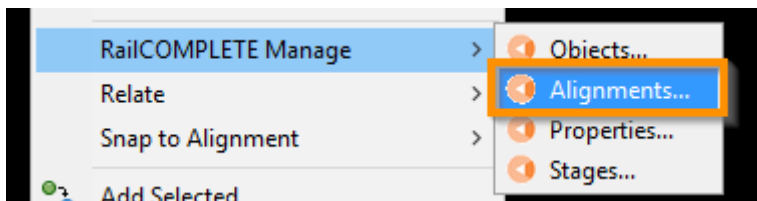
Work with alignments

Define reference alignment

In this example we are going to make the OUTBOUND alignment the parent (reference) alignment for the alignment CROSSING-IN-TO-OUT.



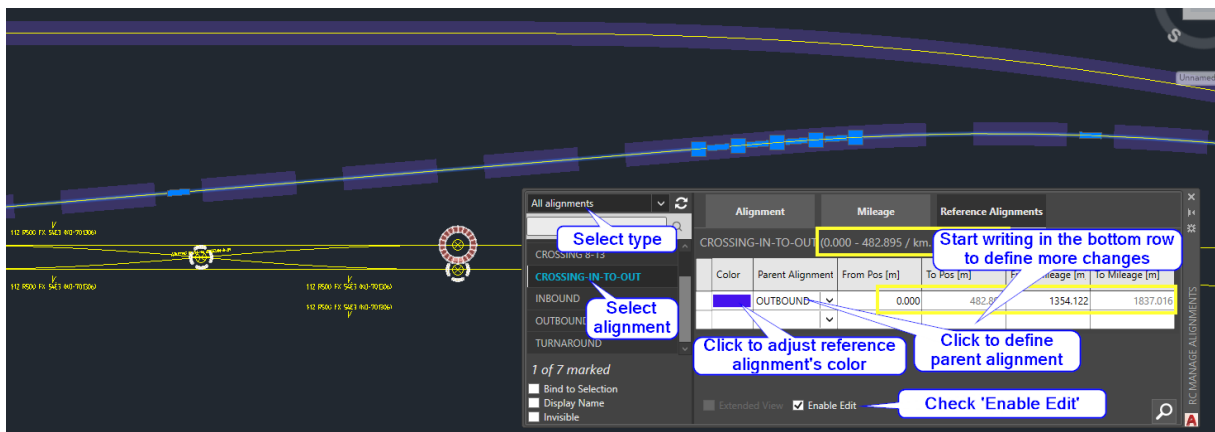
Select the CROSSING-IN-TO-OUT alignment, right click and choose RailCOMPLETE Manage -> Alignments.



This will bring up the Alignment Manager editor, where not only elevation profile and cant profile but also mileage data and reference alignment definitions can be edited.

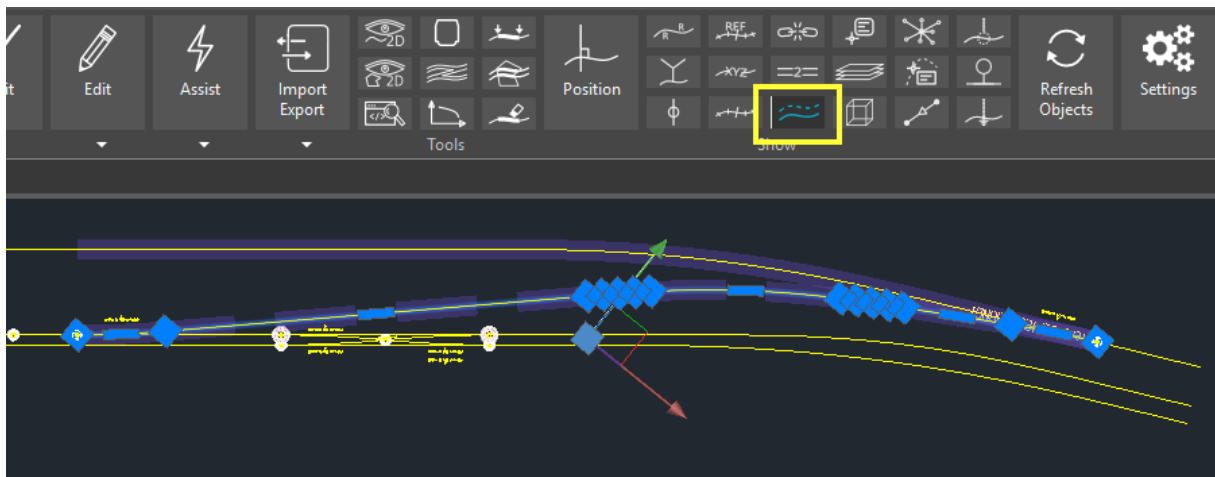
Geometry can only be viewed in the Alignment Manager. Geometries can be adjusted by acting directly on their grips and segment types using the basic CAD system alignment functionality in modelspace.

Select the CROSSING-IN-TO-OUT alignment in the browser section to the left if it didn't show as expected. Activate the Reference Alignments tab, check the 'Enable Edit' box and choose OUTBOUND alignment as its parent alignment.

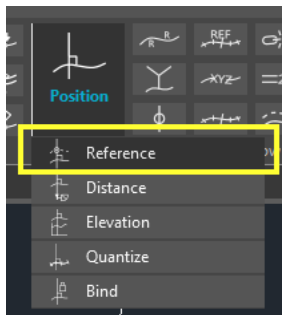


CROSSING-IN-TO-OUT has now inherited mileage data from the parent alignment OUTBOUND, from the start (Pos = 0 / Mileage = 1354.122) of CROSSING-IN-TO-OUT to its end (Pos = 482.895 / Mileage = 1837.016).

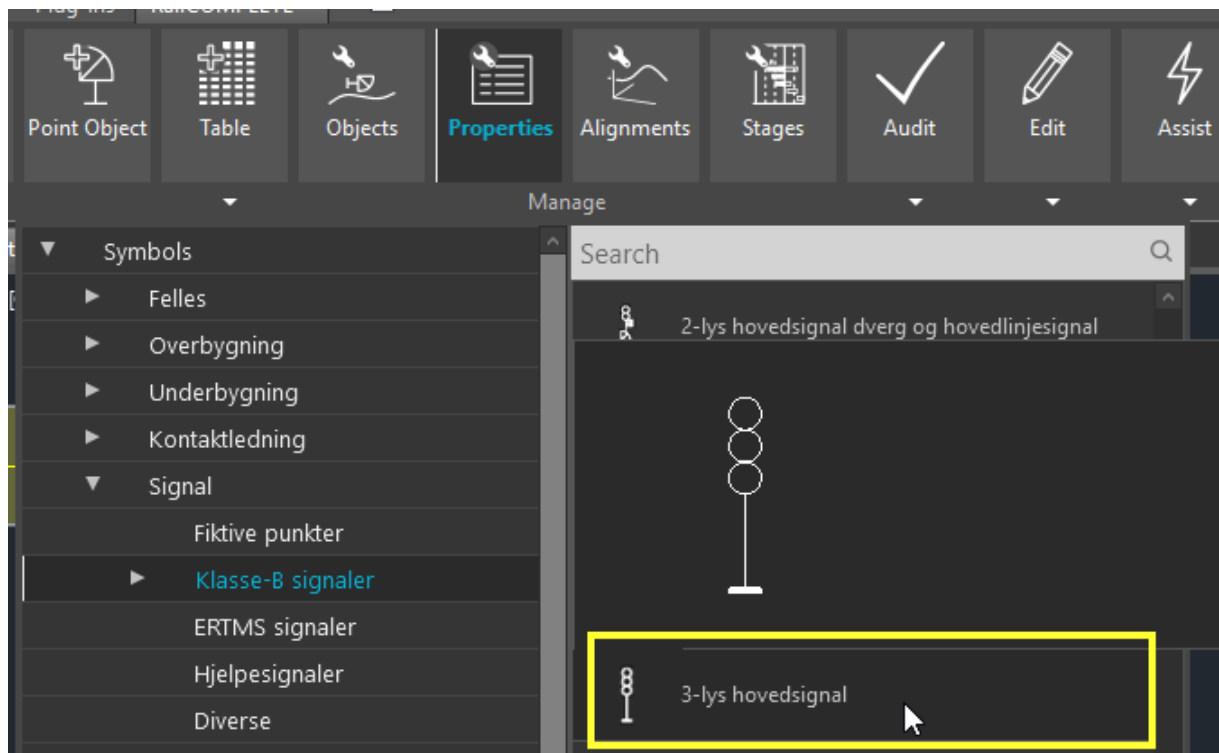
This can be visualized by selecting the CROSSING-IN-TO-OUT alignment and toggle the Show Reference Alignment button in the ribbon.



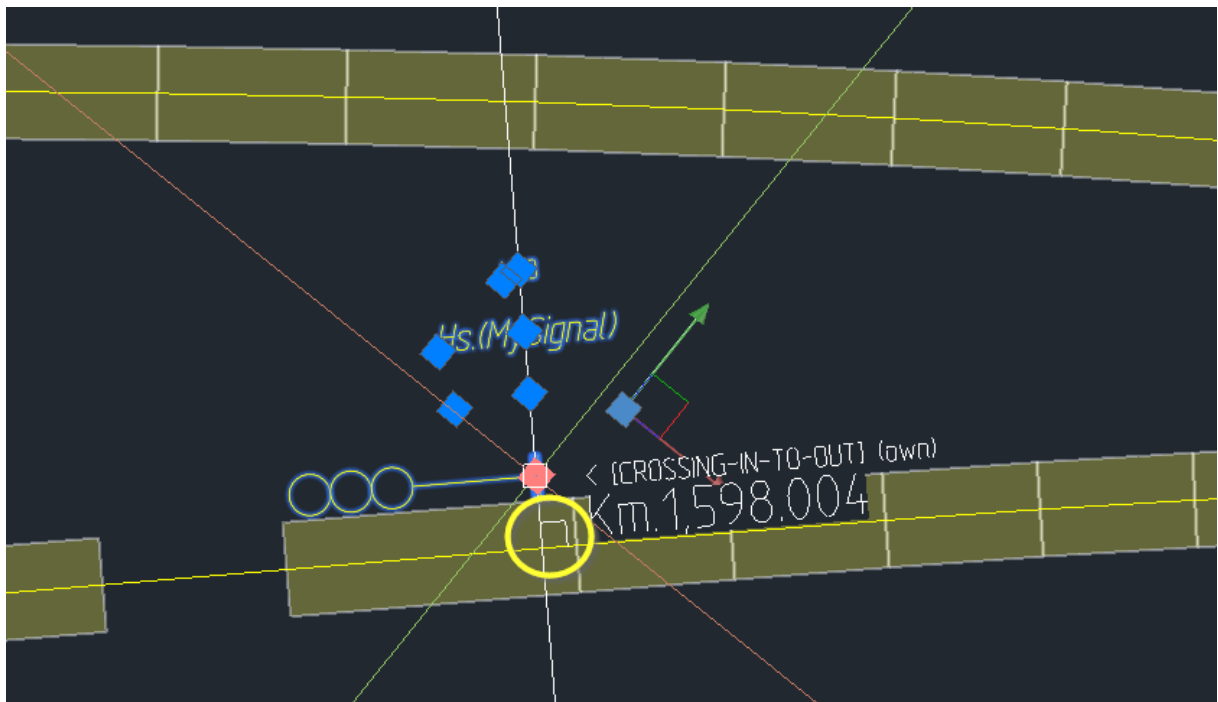
You have now access to two linear positioning systems for objects close to the CROSSING-IN-TO-OUT alignment. Activate the RC-ShowPosition tool using the ribbon button, and make sure that 'Reference' mode is not active.



Consider an optical signal located at mileage 1598.004 on CROSSING-IN-TO-OUT. Use the RC-CreatePointObject button in the ribbon, use the browser or the search tool to locate a Class B optical signal with 3 lanterns and insert it at the suggested default distance from the CROSSING-IN-TO-OUT alignment.



Use the RC-ManageProperties tool to adjust the signal's mileage to exactly 1598.004 as shown below.

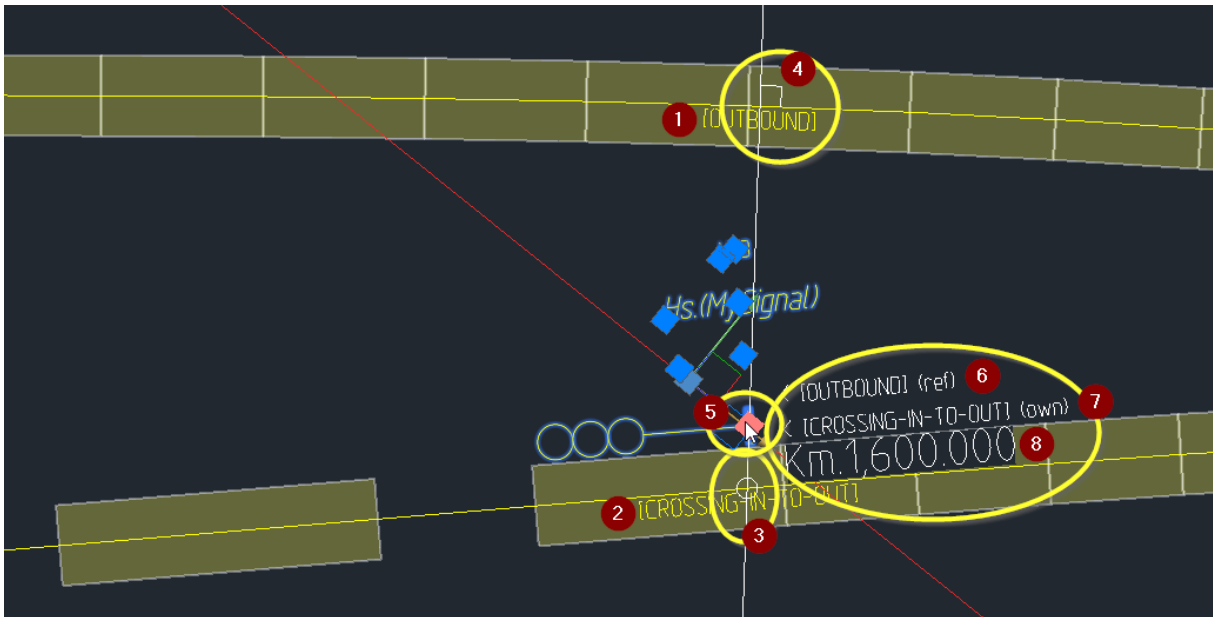


Activate the RC-ShowAlignmentName tool in the ribbon, select the INBOUND and the CROSSING-IN-TO-OUT alignments, zoom and copy the alignment name annotations to the drawing.



Then activate the 'Reference' mode in the RC-ShowPosition tool. Select our signal again.

The parent alignment is shown as a solid line (1) (sometimes shown both as hatched and solid at the same time). The child alignment (the signal's own alignment) is dashed (2). The CAD cursor picks up its position in the alignment where the small circle (3) is shown. Since RC-ShowPosition is now in 'Reference' mode, a small square appears in the reference alignment at the location where the perpendicular passes through the signal's insertion point (5). See that the reported reference alignment name is "OUTBOUND (ref)" (6), the own alignment name is "CROSSING-IN-TO-OUT (own)" (7) and that the shown position, as measured perpendicularly to the OUTBOUND alignment, is 1600.000 (8).

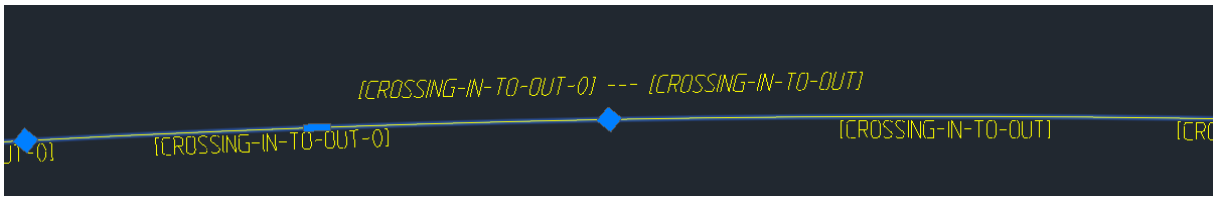


Changing parent alignment

In this chapter, we want to split the CROSSING-IN-TO-OUT alignment into two alignments and assign the TURNAROUND alignment as a reference alignment to the hi-mileage part of CROSSING-IN-TO-OUT.

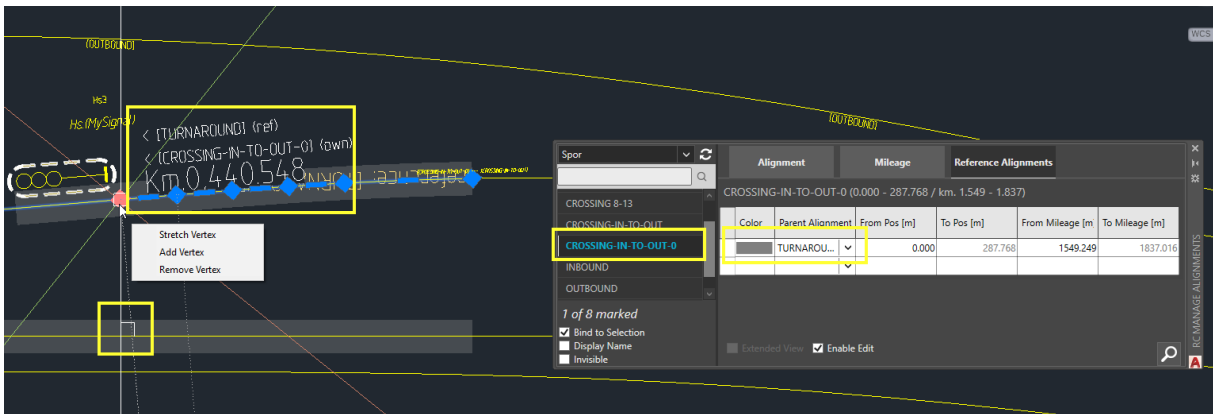
First, we need to split CROSSING-IN-TO-OUT somewhere in the middle. Select RC-BreakAlignment below the ribbon Edit-button or type in RC-BreakAlignment in the model space view.



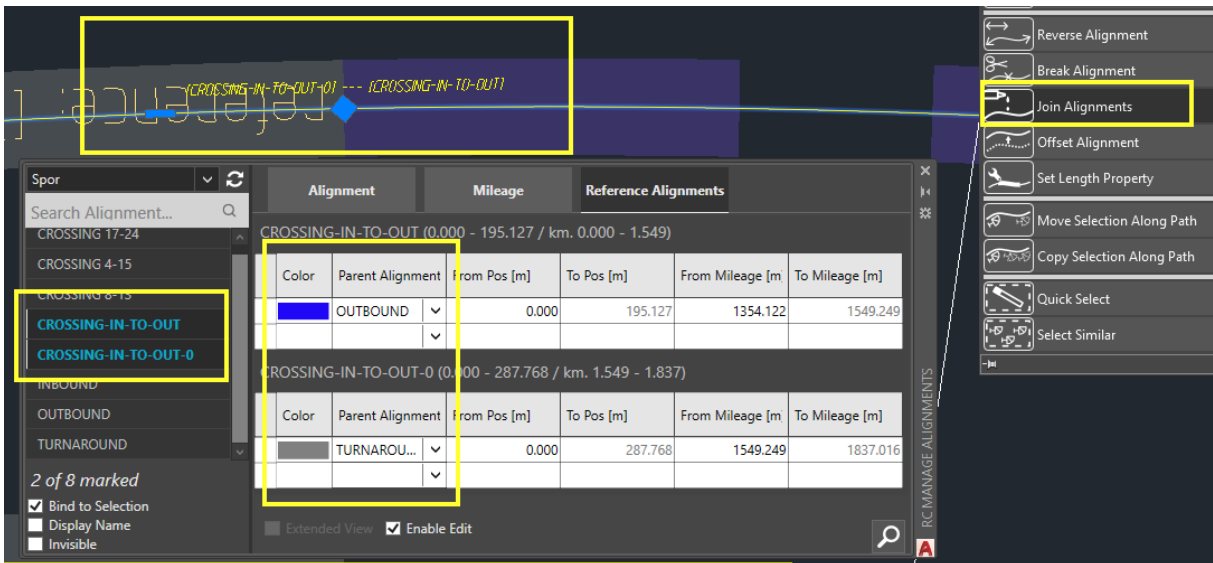


Break the alignment somewhere around reference mileage 1550. The high mileage part has automatically been named “CROSSING-IN-TO-OUT-0”. Notice that an immaterial object has been automatically created, a so-called “continuation” – it links the topology of the tracks together even if we split our track in two halves. The name of the continuation object has automatically been set to “[CROSSING-IN-TO-OUT] --- [CROSSING-IN-TO-OUT-0]”.

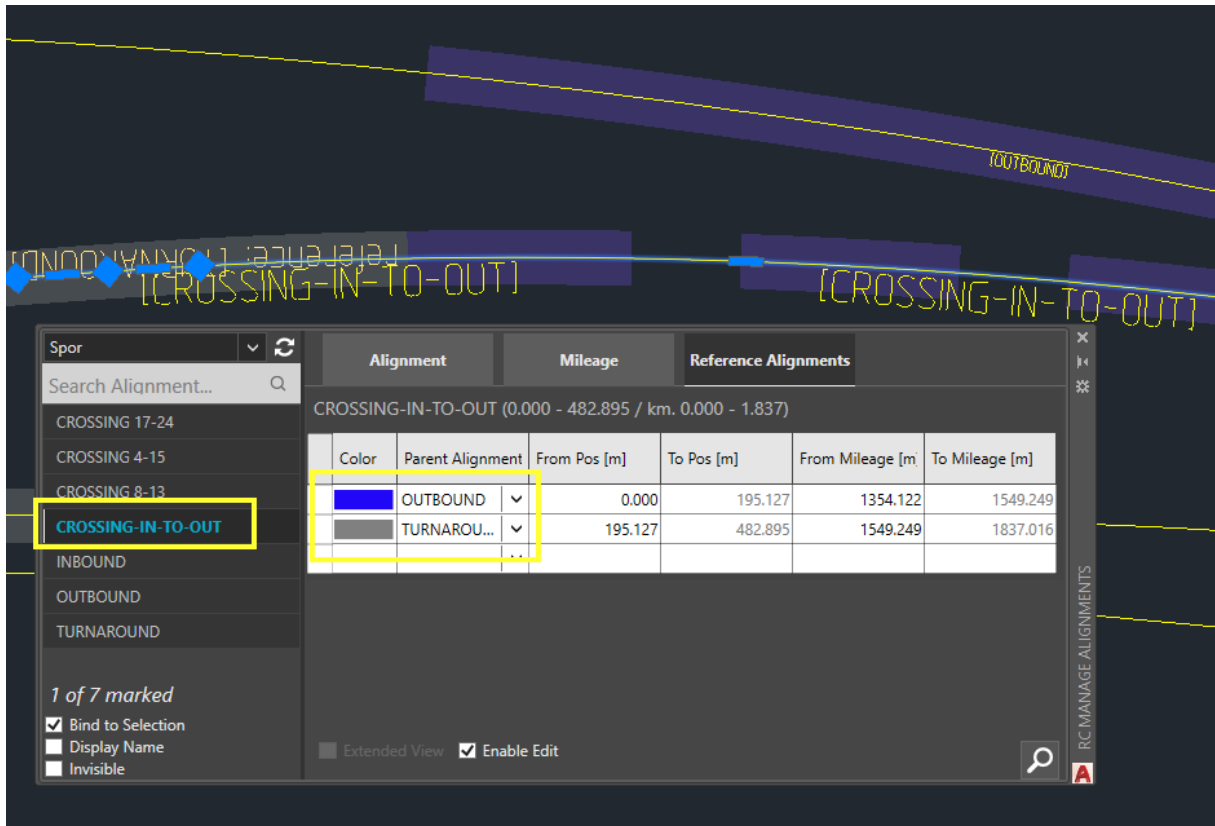
Open the Alignment Manager and locate the new alignment CROSSING-IN-TO-OUT-0. Change its parent alignment as shown below. The TURNAROUND track has a mileage value around 440 at this location, so now the signal has “moved” to location “km 0.440548”.



We can join the two halves again using the RC-JoinAlignments command. This is what things look like before we join...



...and after we join:



The screenshot displays a software interface for managing rail alignments. At the top, a track diagram shows a yellow line representing a track with blue segments labeled 'OUTBOUND' and 'CROSSING-IN-TO-OUT'. Below the diagram is a data grid with the following structure:

Alignment	Mileage	Reference Alignments			
CROSSING-IN-TO-OUT (0.000 - 482.895 / km. 0.000 - 1.837)					
Color	Parent Alignment	From Pos [m]	To Pos [m]	From Mileage [m]	To Mileage [m]
Blue	OUTBOUND	0.000	195.127	1354.122	1549.249
Grey	TURNAROU...	195.127	482.895	1549.249	1837.016

On the left side of the grid, a search list shows 'CROSSING-IN-TO-OUT' highlighted. Below the grid, there are checkboxes for 'Bind to Selection', 'Display Name', and 'Invisible', along with 'Extended View' and 'Enable Edit' options.

Note that we didn't have to split and then join, we could have simply typed in the second row in the datagrid shown above, placing the cursor in the empty second row's "From Pos" column and typed "195.127" etc. The result would have been identical.

Point objects

Point objects are all objects that has a position along the track. This can be signals, signs, switches and masts etc, but also labels, watches and other immaterial objects are considered as point objects.

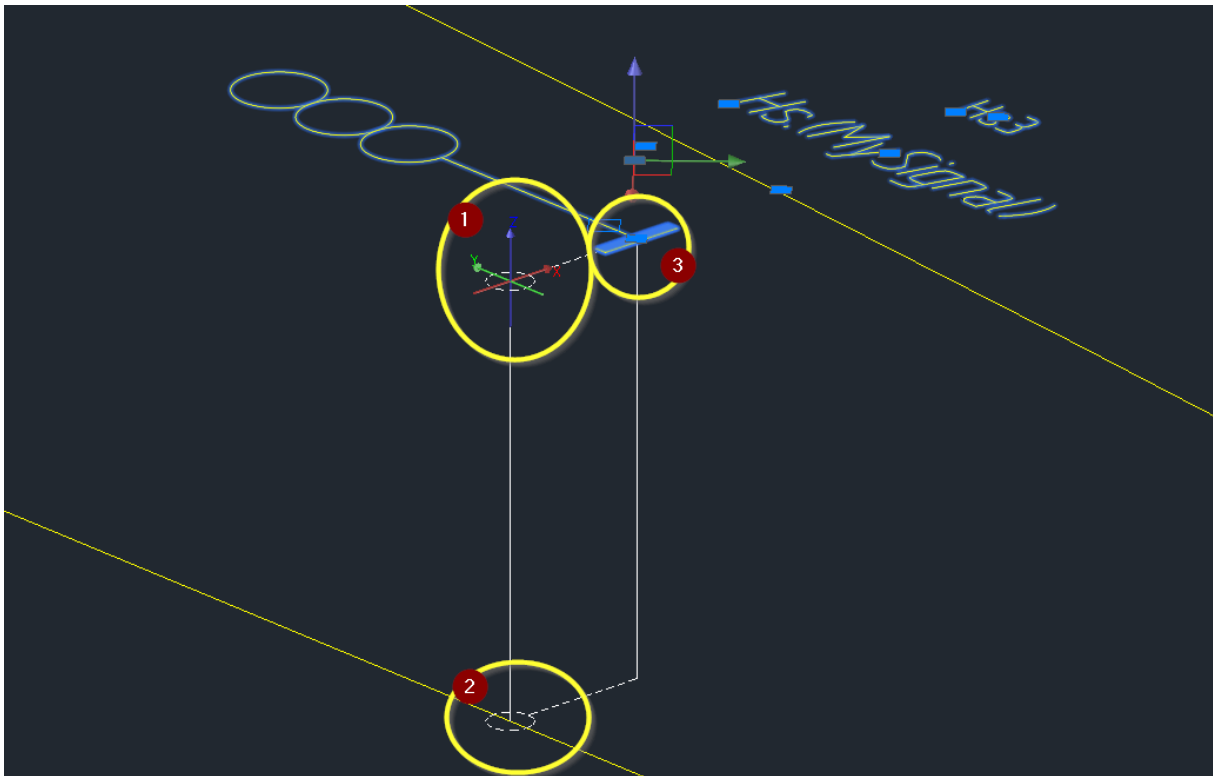
A point object resides in 3D space with a default rotation following the alignment that the object belongs to (the 'own alignment'). Objects follow administration-specific rules for how to orient themselves in space. Some will place themselves at plus or minus 90 degrees rotation around the Z axis relative to the alignment's local tangent direction (such as boards and signals). We tend to call these basic directions for 'up' and 'down' when we speak of signals, boards etc whose main side (the 3D object's front) can only be seen by a train driver in either the increasing mileage direction (up) or in the decreasing mileage direction (down). Other objects might prefer 0 or 180 degrees (such as cabinets and catenary masts). These latter two directions tend to be called 'both' and 'none' because a train driver passing by would either see the object's 3D model's front from both driving directions (cabinet with door facing the track), or from none of them (the door opens away from the track).

Track-bound objects such as balises or axle counters will usually be created with a rotation representing the gradient of the track and the effect of cant (superelevation). For administrations that treat cant as a combined track rotation and track lifting (with $h/2$), the objects will also be lifted with cant.

Rotation about the world Z axis is also called 'yaw'. Rotation about the alignment axis (pointing in the direction of increasing mileage along the local alignment tangent) is called 'roll' (and is generally caused by superelevation). Rotation about the alignment local X axis (which forms a right-handed Cartesian axis system with the Z and Y axes) is called 'pitch' (and is generally caused by a track's gradient). The X axis points horizontally and to the right if you observe from your object's insertion point and look along the alignment in its direction of increasing mileage.

Select the signal again and tilt your drawing into 3D mode – hold down the Shift button and press the middle mouse scrollwheel button at the same time. Move the mouse and you will see your model in 3D perspective. Activate the 'Show Own Alignment' tool from the ribbon.





Locate the tiny colored local alignment coordinate system (ACS) axes (1). It features a red X-axis, a green Y-axis and a blue Z-axis. The origo is placed on the own alignment axis and such that there is a plane perpendicular to the alignment's tangent containing the object's insertion point.

If you follow the vertical line from the ACS' origo down to 0 elevation above sea level (2) you will find a dashed circle indicating where in the XY-plane projection of the alignment (i.e., the geometry before profile data is applied) the ACS is located. Extending from the alignment and sideways you move along a dashed line to a point directly below the object's insertion point. Going upwards from here to the object's insertion point, you arrive at the object's elevation above sea level (3), which may be above, level, or below the own alignment's local elevation.

The Y coordinate denotes an offsets from the object's own mileage. The X coordinate denotes a distance sideways from object to own alignment, disregarding elevation differences. The Z coordinate is the object's relative elevation, i.e. its elevation above sea level minus the elevation of its own alignment's alignment-axis-elevation. Thus, the object is located at ACS location (distanceToAlignment, 0, relativeElevation). ID-boards or other artefacts that may be attached to a signal may have other ACS coordinates.

A Y-offset from an object's insertion point may also be called "offset towards driver" and represents a positive or a negative absolute local Y offset, but depending on the object's up/down direction and the direction that the driver is running in.

Positive sideways offsets from an object's insertion point towards its ACS origo are called "offset towards alignment", i.e. the X-offset is negative if the objects is located to the right of its own alignment, and positive if the object is located to the left of its alignments. Please check for yourself.

Aligned object	
Dir	up
Alignment	[CROSSING-IN-TO-OUT]
Reference alignment	[TURNAROUND]
Mileage	1598.004
Reference Mileage	442.501
Side of alignment	Right
Distance to alignment	3,500
Relative elevation	f 0.000
Geographical data	
Relations	
3D	
3D Offset	0.000 0.000 0.000
X	0.000
Y	0.000
Z	0.000
3D Rotation	0.000 0.000 0.000
3D Pitch	0.000
3D Roll	0.000
3D Yaw	0.000
3D Annotations	{}
3D Pole Routing	None
3D Model Name	NO-BN-3D-SA-SIG-NSI63-HS-3-LYS
3D Offset Towards Driver	0.000
3D Offset Towards Alignment	0.000

In the case of an axle counter, the Z-axis rotation (yaw) is either +90 (left rail) or -90 degrees (right rail), since the 3D object's 'front' is considered to be the sensor side facing the centre of the track. The Pitch and Roll values are due to local gradient and cant. The Relative Elevation of 22 millimeters is due to the lifting effect of local cant, the axle counter being mounted in the outer rail.

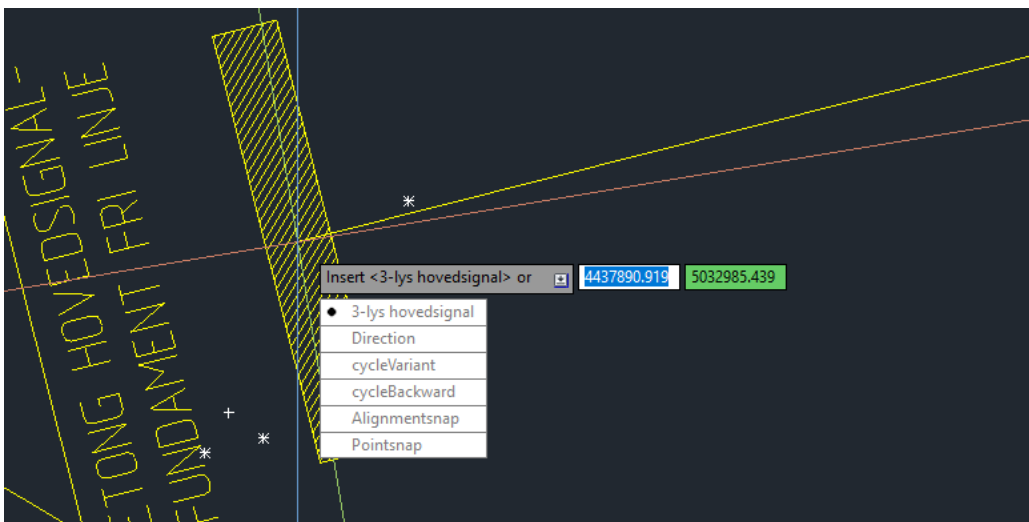
The 'Geo coord' row always shows the CAD system's World coordinate system XYZ values (Easting, Northing, Elevation).

Aligned object	
Alignment	[CROSSING-IN-TO-OUT]
Reference alignment	[TURNAROUND]
Mileage	1574.701
Reference Mileage	419.029
Side of alignment	RightRail
Distance to alignment	<i>f</i> 0.750
Relative elevation	<i>f</i> 0.022
Geographical data	
Geo coord	305053.879 6703212.368 9.488
Relations	
3D	
3D Offset	0.000 0.000 0.000
3D Rotation	↓ <i>f</i> 0.716 0.826 -90.000 <i>f</i>
3D Pitch	<i>f</i> 0.716
3D Roll	<i>f</i> 0.826
3D Yaw	<i>f</i> -90.000
3D Annotations	{}
3D Pole Routing	None
3D Model Name	NO-BN-3D-SA-TEL-THALES-TELLEPUNKT-SK30K
3D Offset Towards Driver	0.000
3D Offset Towards Alignment	0.000

Insert object

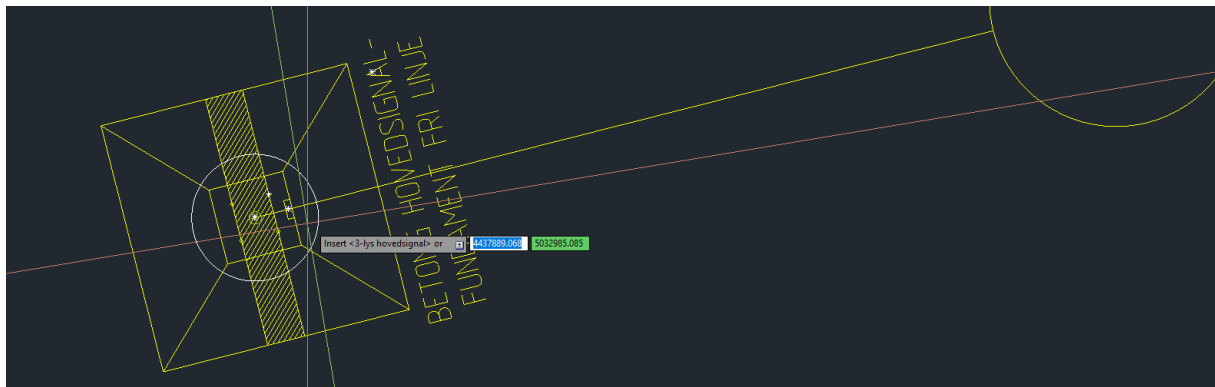
To insert a signal, open the RC-CreatePointObject menu from the ribbon and search for any signal (see the previous tutorial chapter, where we inserted a 3-lantern signal). Click on the desired signal and then hover it over the desired target area.

The object will snap to a default administration-dependent distance to own alignment, which is typically +/- 3.5 m. 'Alignment Snapping' can be toggled during the insertion process, as well as 'Point Snapping'. The Point Snapping mode is useful if there is an existing object nearby that you would like to dock into, for instance a signal that you would like to dock to a signal foundation, or vice versa – depending on what came first.

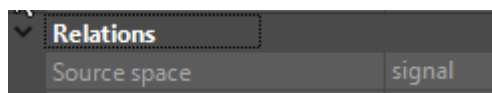


Use shortcut keys D, V, B, A and S to flip directions up/down, next/previous object variant, and toggle alignment/point snap mode.

In the example below we have first created a signal foundation, then we inserted a signal using the 'Point snap' option. Notice the white circle around the offered snapping point (centre of foundation), as well as the fact that the signal's base plate has snapped to the foundation's centre even if the CAD cursor is still hovering somewhat to the side of the foundation's centre.

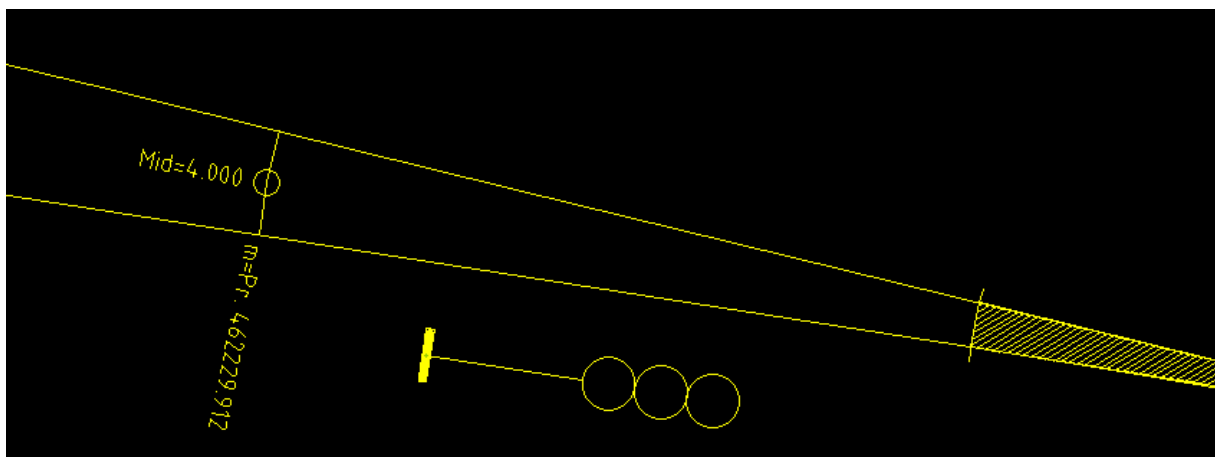


Point snapping can be tailored in the snap-point offering object's DNA to be sensitive to all incoming object types, or just one or a few selected ones. Each object type has a 'relation space' declaration in its DNA which identifies the type of objects it may snap to or offer snapping to. Alignment snapping can also be made dependent on the target alignment's relation space name. Typically, tracks' space is called "spor" (Norwegian for 'track'), signals' space is called "signal" etc. One object type's source space becomes another object type's target space.



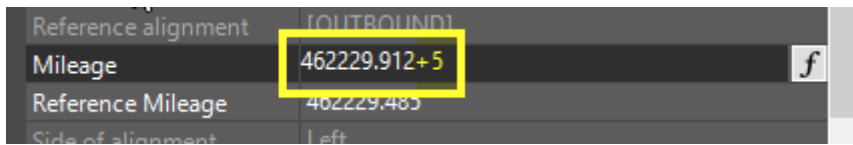
Positioning the signal

In this example we want to position the signal 5 meters ahead of the closest fouling point.

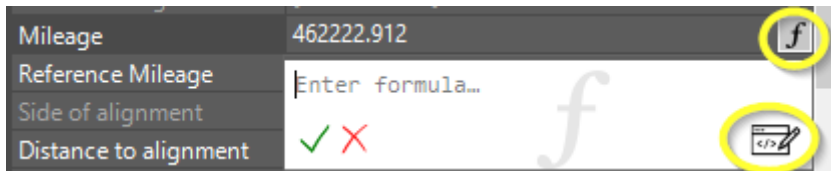


The signal has been placed carelessly close to a switch.

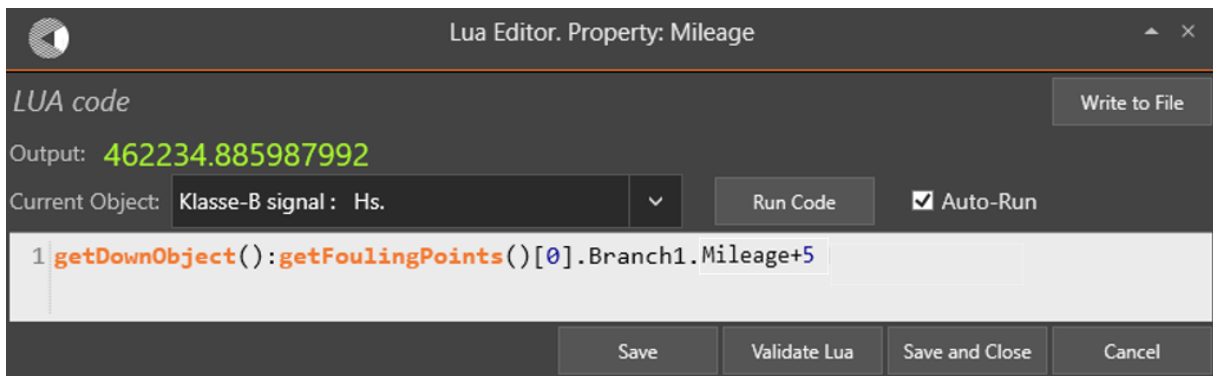
One way to position the signal correctly is to first place it directly outside the fouling point marker (which here has a mileage equal to 462229.912). Activate RC-ManageProperties and add 5 meters to the signal's mileage property. Just enter "+5" after the mileage shown and press ENTER:



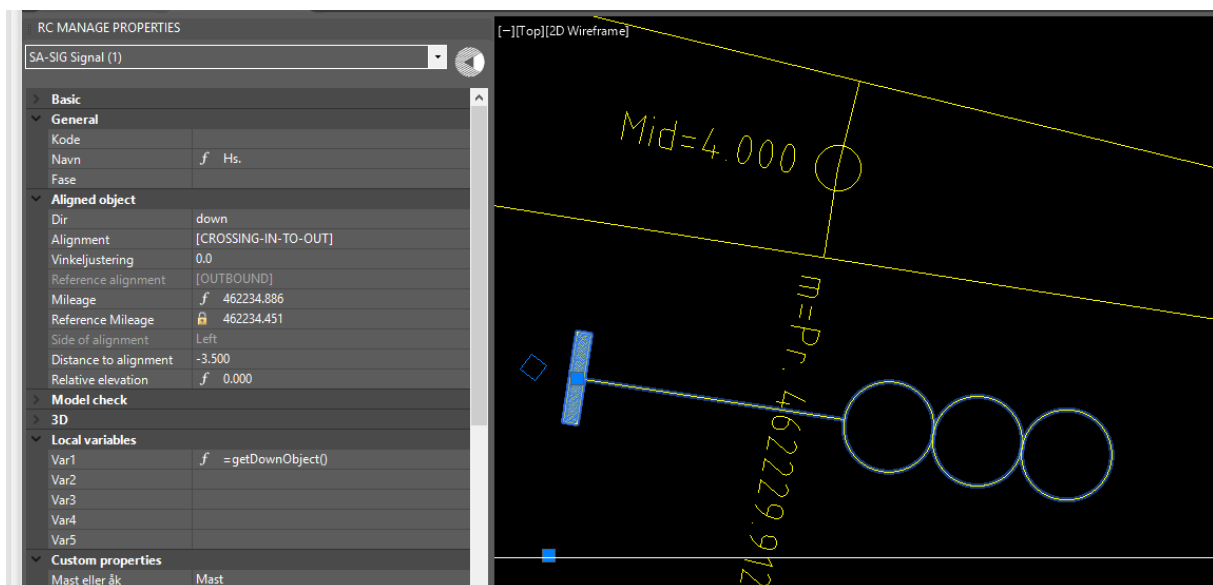
Another option is to add a Lua formula directly into the mileage property. Lua is the scripting language that is used everywhere within RailCOMPLETE. First click the mileage property row, then click the 'f' (formula), then click the editor symbol.



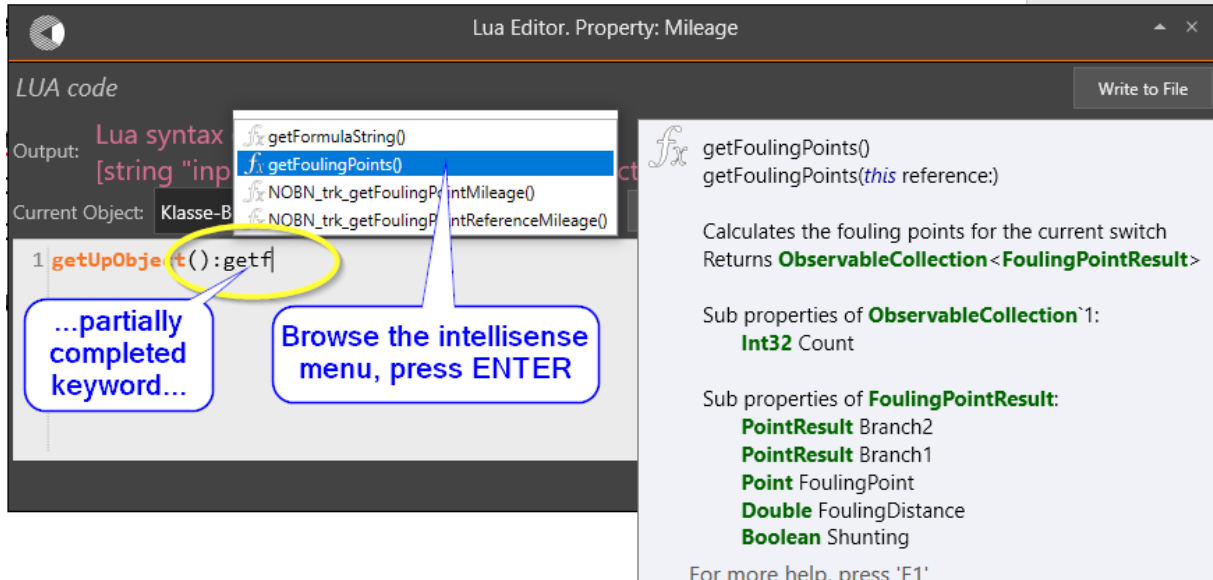
Alternatively, just place your cursor in the property's row and press function key 'F3' (Lua editor).



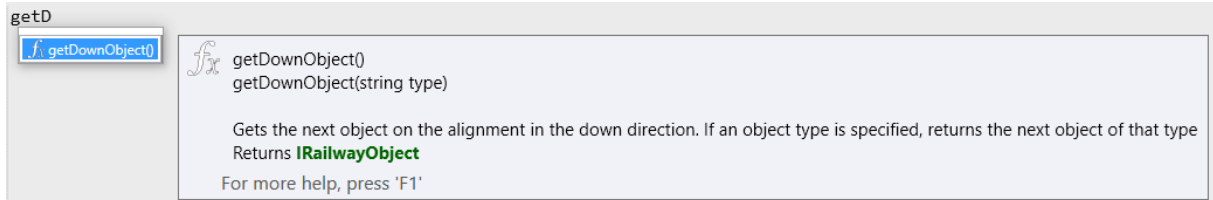
The formula above (which is pretty advanced for most users) asks for the closest fouling point in the down direction and extracts its mileage value and adds 5 meters. This formula actually works only if the closest object is the switch and not an axle counter or something else. To make sure that a switch is found, you could use `getDownObject("KO-SPV Sporveksel")` to ignore any objects other than (Norwegian) switch objects. The result should like this:



It is easy to create custom formulas. Intellisense guides you, auto-completing keywords as you type. The output value from your formula is constantly validated, unless you turn the Lua-editor's Auto-Run feature off. Hovering over a built-in function's name brings up a popup window with brief information about call syntax and arguments. Function key 'F1' brings you to the online Help system where more information is to be found.



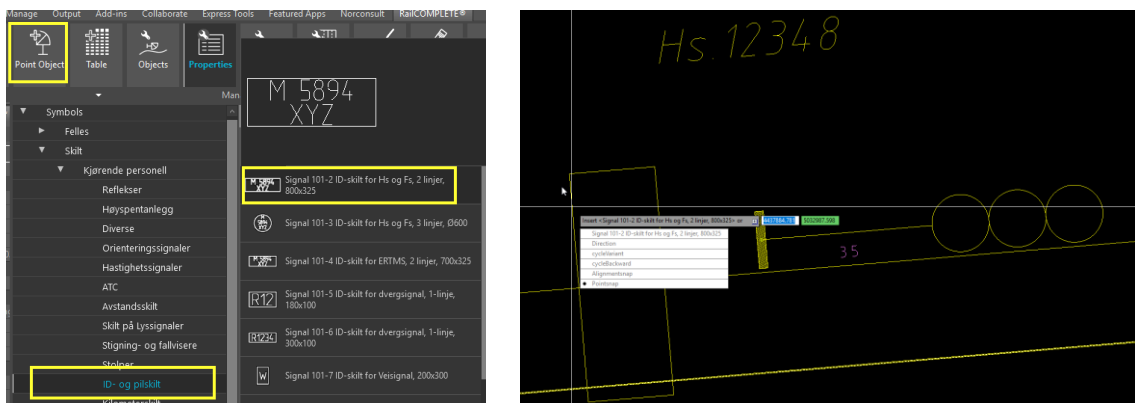
See more on Lua programming in a chapter further down.



Attaching a point object to another point object – making a family

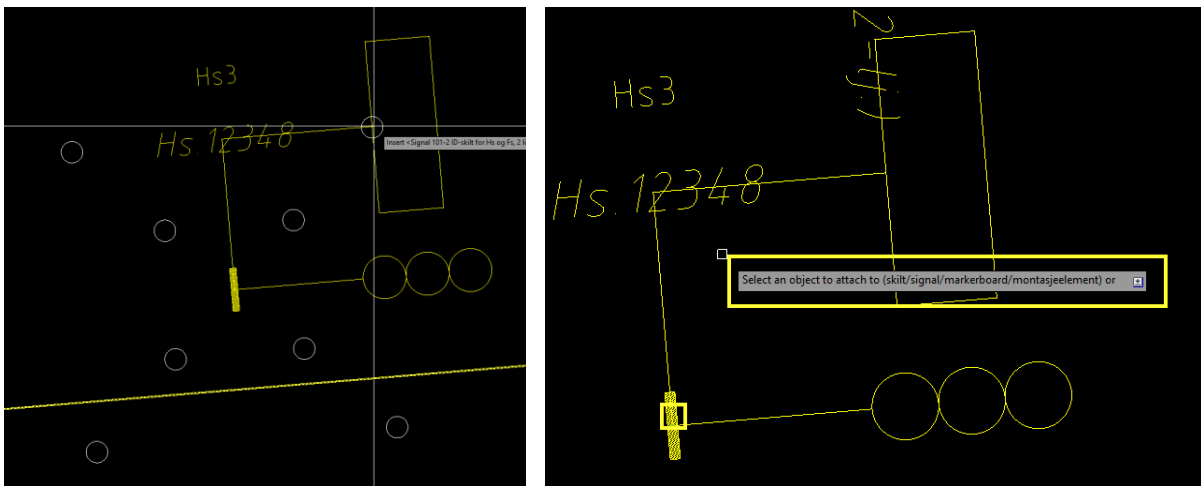
The term 'attach' is used in RailCOMPLETE with a specific meaning. It means that the attached object, the "child" object, is forced to inherit the insertion point coordinates from its "parent" object.

As an example, insert an ID board ("Skilt" in Norwegian). While hovering with mouse close to a signal, waiting to place the ID-board, press 'd' to change direction (if needed), and press 'p' to engage point snapping (if needed).




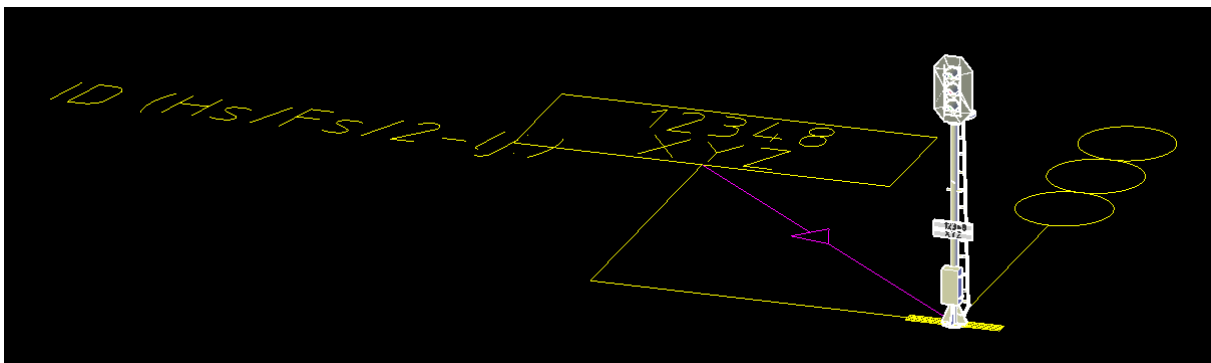
The signal offers a small circle around its insertion point – move your mouse to this point and click somewhere inside the circle, and the ID board’s insertion point snaps into place.

You are then asked to click once more, in order to place the board’s 2D symbol at a suitable place nearby the signal. This location is called the symbol’s “Display Point”. The DisplayPoint always resides at the same elevation as the object’s insertion point. The 2D offset from the object’s insertion point to its displaypoint is called the “Symbol Offset”. A “tail” will automatically be drawn from the object’s displaypoint to its insertion point. The tail ‘wiggles’ in turns of 90 degrees and in such a way that the tail implicitly shows you the object’s direction and side of alignment. The tail starts from the insertion point in the direction which is perpendicular to the object’s alignment and heading away from the alignment. The tail ends in the object’s displaypoint and heading in the object’s direction.



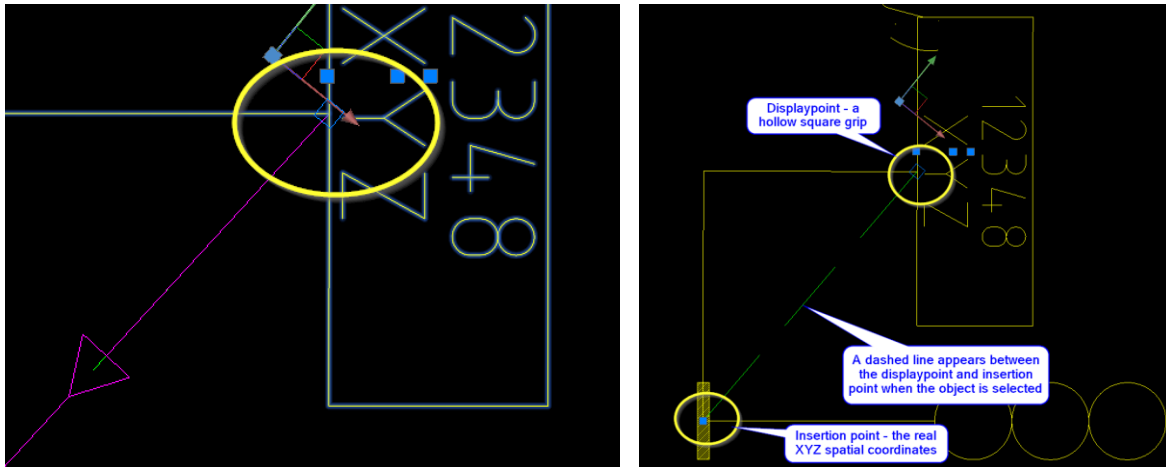
In the last object insertion step you will be asked to attach the ID board (the child) to a suitable parent object. A choice of object spaces is shown, “skilt/signal/markerboard/montasjeelement”. Click on the signal to execute. They now form a tiny “family”, with the signal being the head of the family.

Now, turn on the annotation tool for attachments, RC-ShowAttachmentLines . Select the signal or the ID board and see that the parent-child relationship is shown graphically with a directed arrow symbol from child to parent. See what it looks like with also the 3D annotations copied to the drawing – and notice that the ID board has automatically received labeling texts from its parent:



Quite often, your drawings will become crowded with objects having 2D symbols that overlap (since 2D symbol sizes are usually quite large compared to their actual physical size). Consider an ID board

symbol which should be moved. Simply drag and drop the ID board's graphics using the small square shown at the displaypoint³.



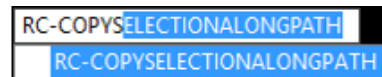
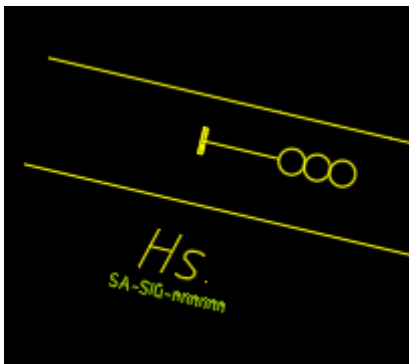
Even for objects such as signals you might want to displace one or both signals when the drawing gets messy. Signals placed back-to-back on the same side of a track looks like this before and after cleaning up the graphics:

Most object types have symbols with displaypoints which are allowed to be moved away from the insertion point, but this is to be decided in the DNA for each object type.

Copy point objects

You will often need to duplicate an object or a group of objects along an alignment. In this example we will copy a signal four times in increments of 35 meter.

Create a signal, place it at least 150 meters from the end of an alignment.



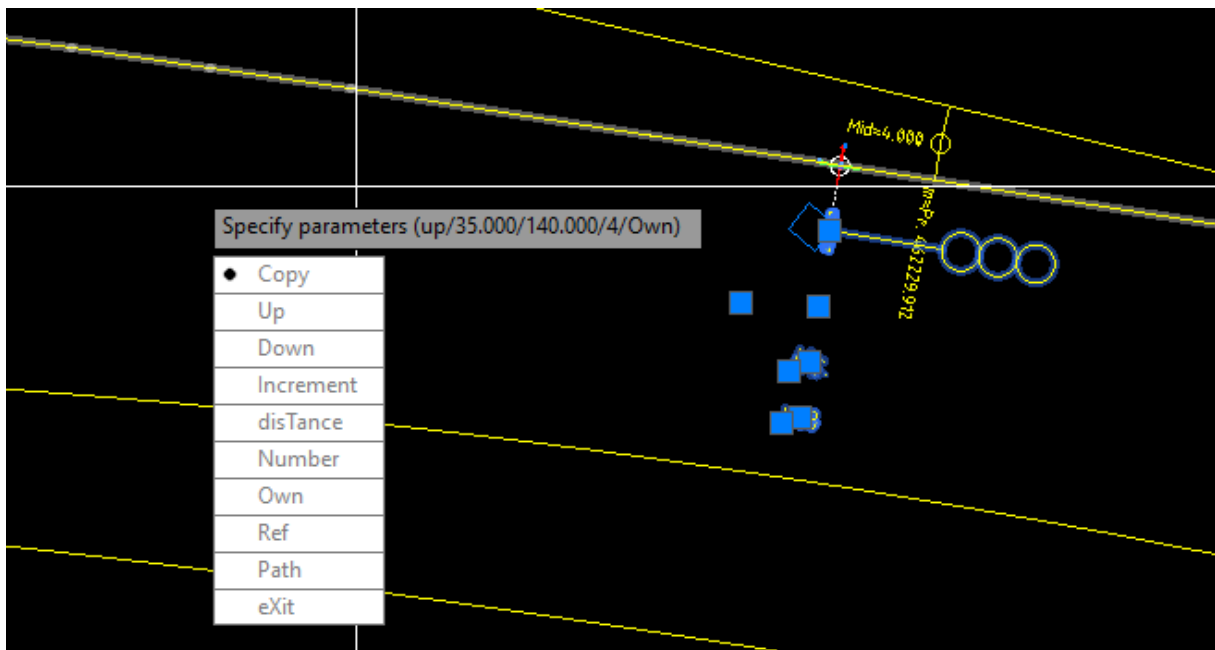
Start the command RC-CopySelectionAlongPath from the menu (below the Edit button) or start it by typing in the command name.

³ If the displaypoint is identical to the insertion point (such as a signal, a switch), then the displaypoint's square grip will appear a little sideways from the insertion point. If the displaypoint has a non-zero symbol offset (such as a board or a section symbol), then the grip appears at the display point.

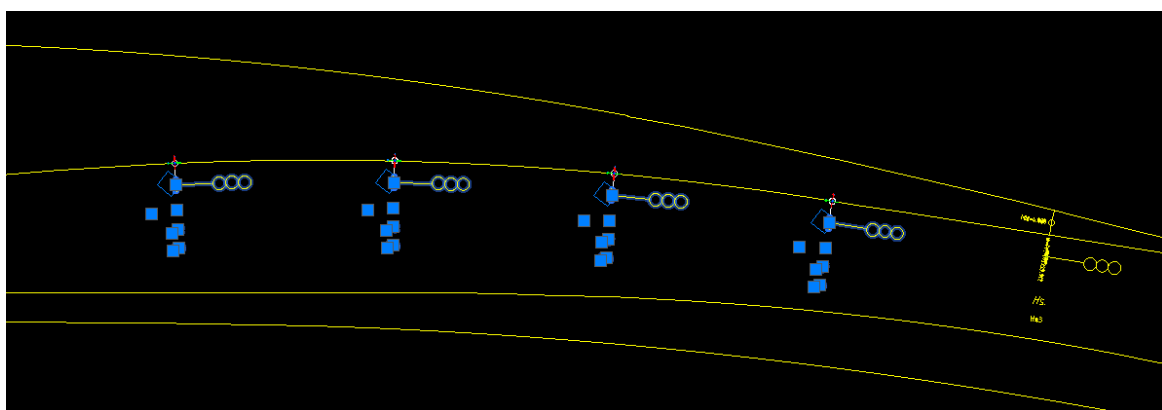
Follow the instructions and set ‘Up’ as copy direction, use 35 meters as ‘Increment’ and set ‘Number’ to 4 before selecting ‘Copy’. The current parameter selections are shown in the command prompt – direction / increment between copies / total distance from source selection to where the last copy can be located / number of copies to produce / measure increments along own or reference alignment.

Since the three parameters Increment / disTance / Number are interconnected, entering a change in one of them will modify the oldest of them accordingly and keep the “middle one” unchanged.

If needed, you may switch into Path mode, where RailCOMPLETE will stop and ask for directions (left/right) for each connection object where there is a choice of paths to continue along.

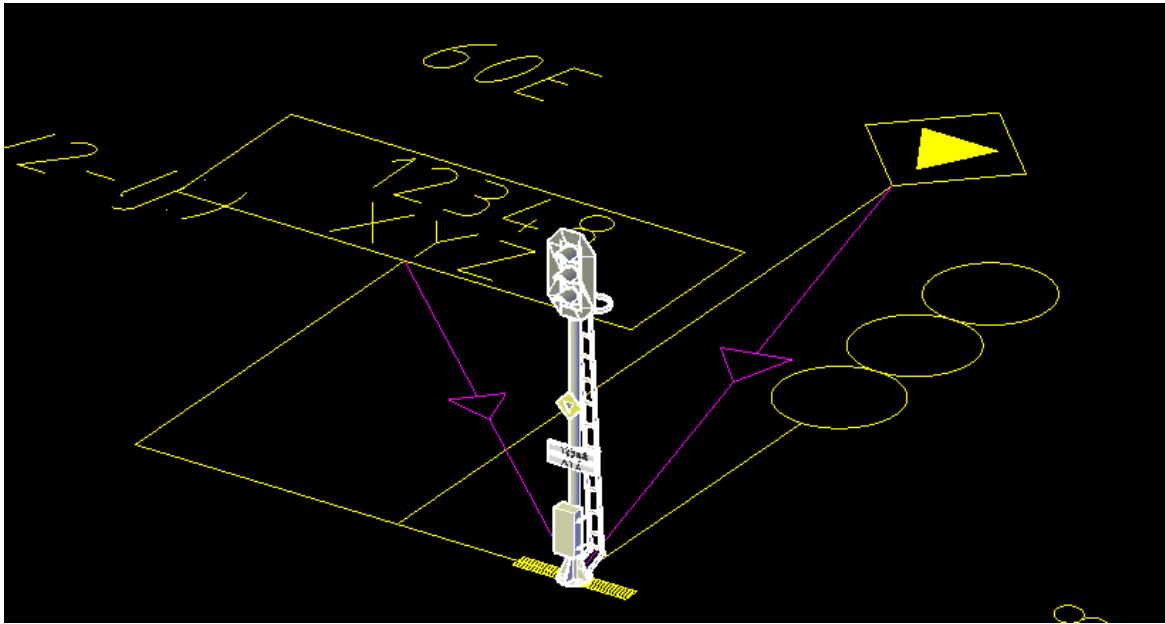


The result may look like this:

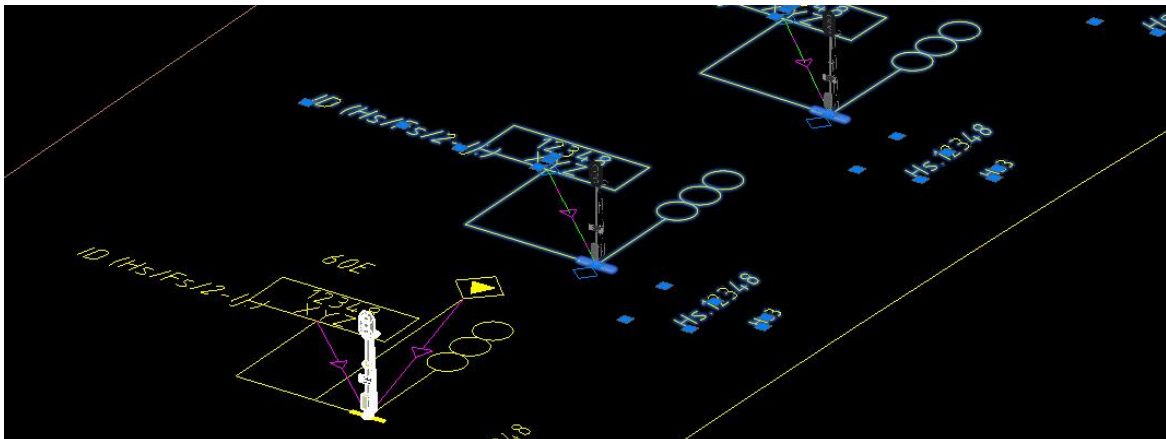


If there are attachments (or relations, see more about relations further down) internally between objects in the source object group, then these will be preserved also in the copies.

As an example, consider a signal with an ID board (a white plate with texts) and an ATP warning board (the losange with a down-triangle), both are attached to the signal.



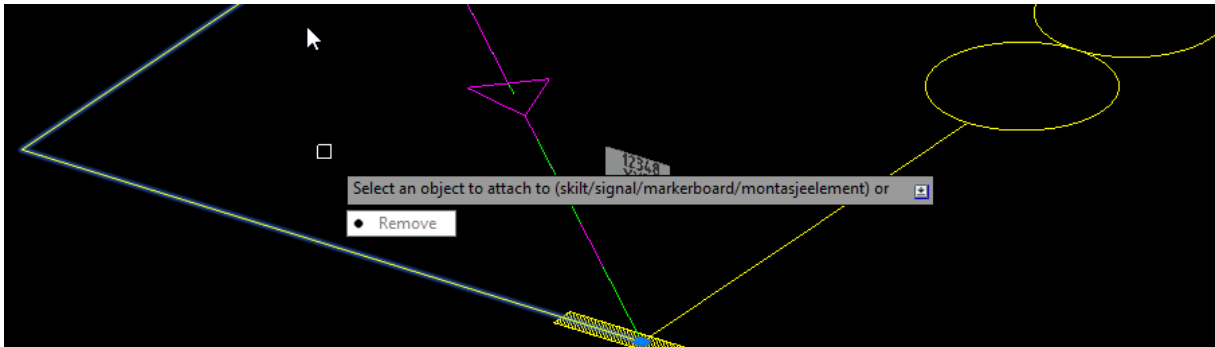
Select just the ID board and the signal and copy these two objects using RC-CopySelectionAlongPath:



You may at any moment split up a family by breaking a parent-child attachment. Select the child object, right-click and select “Attach to <list of allowed parent attachment spaces that you may attach the child to to>” from the context menu.



Either click on the desired parent object (which must belong to a compatible object attachment category), or press 'R' to remove.



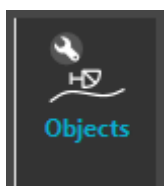
Attachments may be nested to form a family tree.

Managing Objects

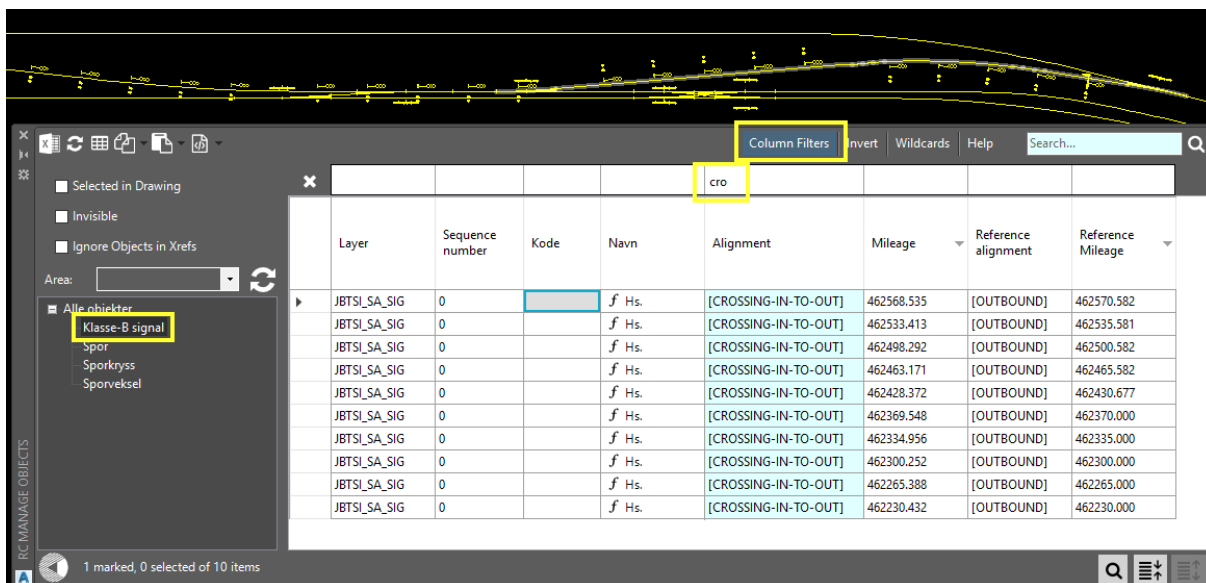
The object manager is a versatile tool for searching and filtering, find and replace, mass update, to locate objects in the drawing and just get a good overview of your database contents – i.e., your drawing(s).

Autonumbering

This example shows how to autonumber and autaname signals along the INBOUND alignment. Start by placing a couple class-B (optical) signals of along this alignment, then open the Object Manager. You may start it using the RailCOMPLETE Manage Objects ribbon button, or the RC-ManageObject command, or start it from the context menu Manage -> Objects... or from the Command Browser.



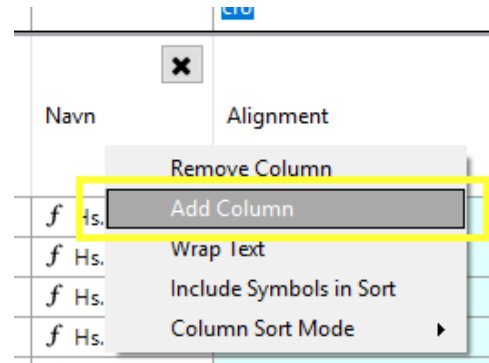
Show the class-B signals belonging to alignment CROSSING-IN-TO-OUT. You can do so for instance by using a combination of Column Filters (top row) and the object type browser (left).



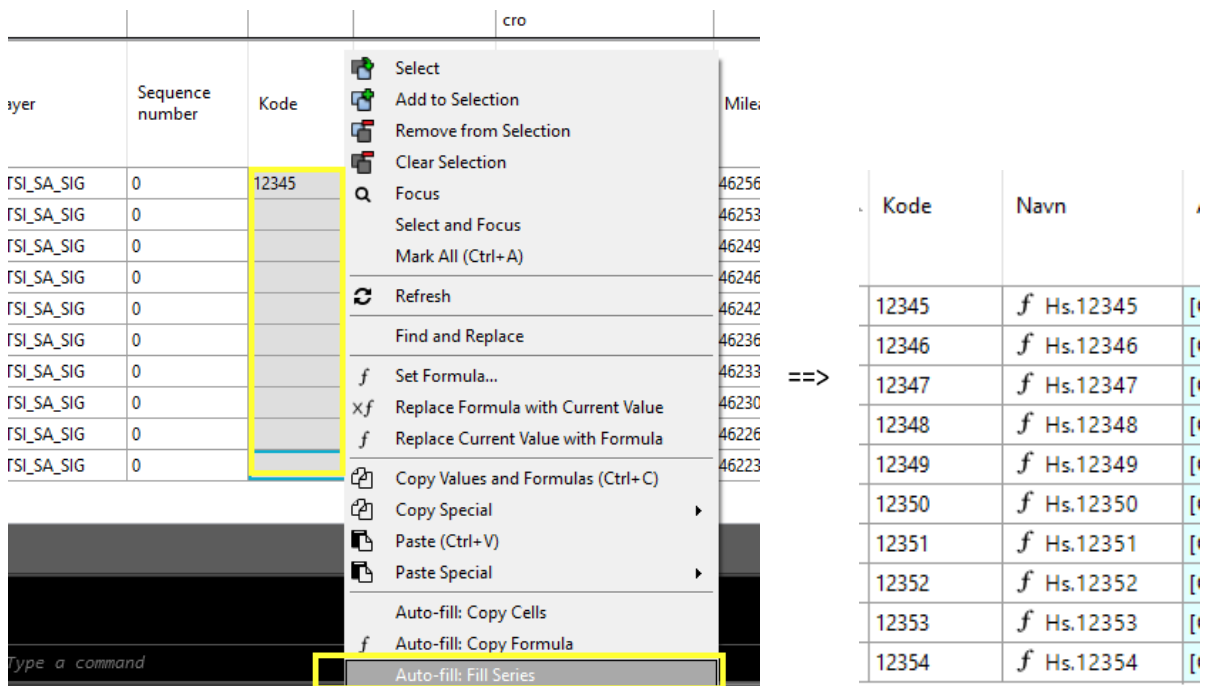
Layer	Sequence number	Kode	Navn	Alignment	Mileage	Reference alignment	Reference Mileage
JBTSI_SA_SIG	0		f Hs.	[CROSSING-IN-TO-OUT]	462568.535	[OUTBOUND]	462570.582
JBTSI_SA_SIG	0		f Hs.	[CROSSING-IN-TO-OUT]	462533.413	[OUTBOUND]	462535.581
JBTSI_SA_SIG	0		f Hs.	[CROSSING-IN-TO-OUT]	462498.292	[OUTBOUND]	462500.582
JBTSI_SA_SIG	0		f Hs.	[CROSSING-IN-TO-OUT]	462463.171	[OUTBOUND]	462465.582
JBTSI_SA_SIG	0		f Hs.	[CROSSING-IN-TO-OUT]	462428.372	[OUTBOUND]	462430.677
JBTSI_SA_SIG	0		f Hs.	[CROSSING-IN-TO-OUT]	462369.548	[OUTBOUND]	462370.000
JBTSI_SA_SIG	0		f Hs.	[CROSSING-IN-TO-OUT]	462334.956	[OUTBOUND]	462335.000
JBTSI_SA_SIG	0		f Hs.	[CROSSING-IN-TO-OUT]	462300.252	[OUTBOUND]	462300.000
JBTSI_SA_SIG	0		f Hs.	[CROSSING-IN-TO-OUT]	462265.388	[OUTBOUND]	462265.000
JBTSI_SA_SIG	0		f Hs.	[CROSSING-IN-TO-OUT]	462230.432	[OUTBOUND]	462230.000

Enter the number 12345 to the first cell in the 'Kode' (code) column and auto-number the objects' code property. Start with marking all the 'Kode' column cells. To mark cells, use Excel-style keyboard shortcuts (Click, Ctrl+Click, Shift+Click, Click-hold & sweep, Ctrl+A etc), the marked cells will turn grey.

If the 'Kode' column is not visible (it might be called 'code' in your installation), add the 'Kode' column by right-clicking the column header section where you want to add the 'Kode' column and select 'Add column'. Make sure that the Alignment and Name columns are also visible.



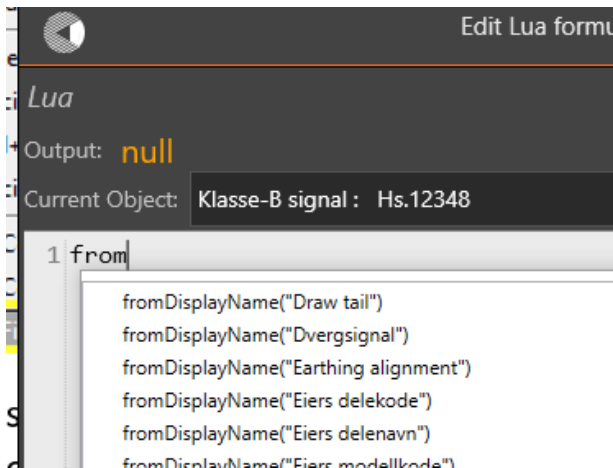
Then, right-click somewhere in the data grid (in one of the cells holding a value) to bring up the context menu and select 'Auto-fill: Fill Series'.



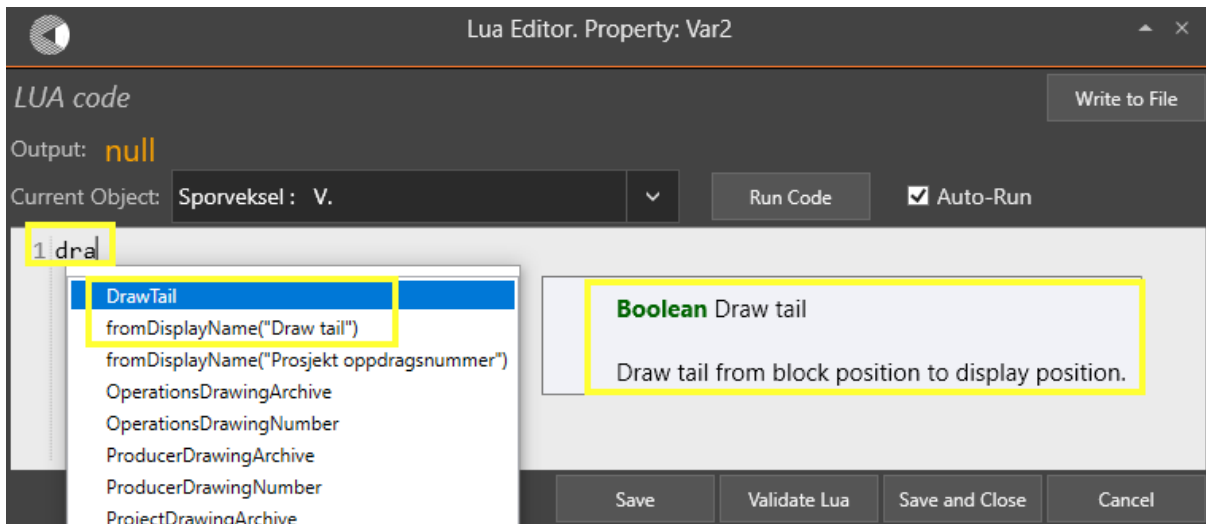
Note how the name ('Navn' column) also is updated accordingly. This is because the name column by default has a built in formula '=Hs.."code' or similar. See also the online Help documentation (F1).

Property names versus property displaynames

Please note that as a general principle, properties in RailCOMPLETE will have two names. "Real" property names (which are used in the database searches in RailCOMPLETE) are by convention written in English, whereas so-called "displaynames" are written in the administration's local tongue. As an example, the catenary cantilever custom property "Pushdirection" has Norwegian displayname "Kraftretning", and so on. What you see in the Property Manager and in the Object Manager datagrids will be the displaynames, but in Lua formulas you can choose between using the real name, or using the displayname. To reference a property by its displayname, use the built-in formula construct 'fromDisplayName("My property's displayname is here")'.



If no explicit displayname has been assigned to a property (in the DNA), then the property's name is also used as a basis for its displayname. 'PascalCased' and 'camelCased' real names are expanded into a word sequence – 'DrawTail' becomes "Draw tail", and so on.

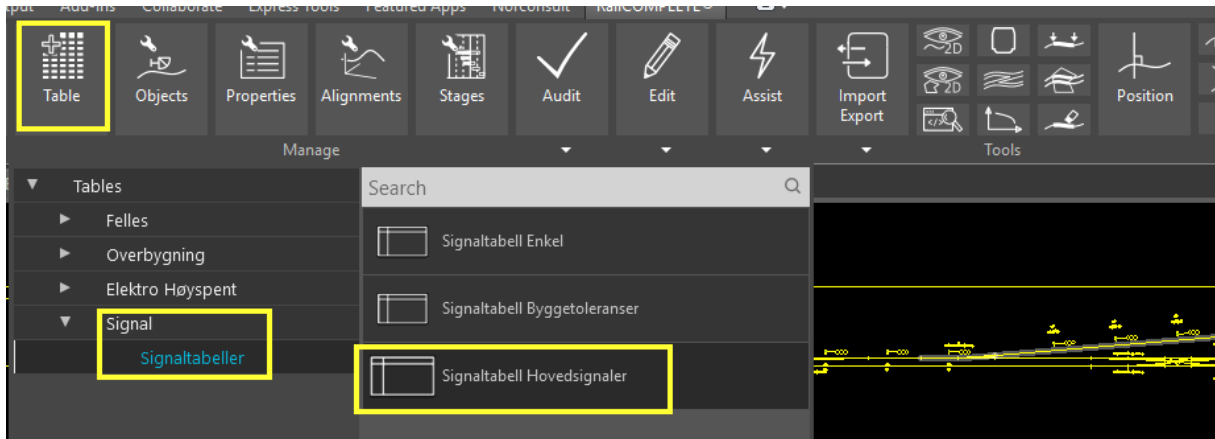


Tables

RailCOMPLETE has a built-in table feature with predefined 'live' tables that enables users to update values whenever changes are done in the drawing.

Create predefined table

Select a predefined table below the Table button. Select a predefined signal table and then click in CAD modelspace to place it somewhere in the drawing.

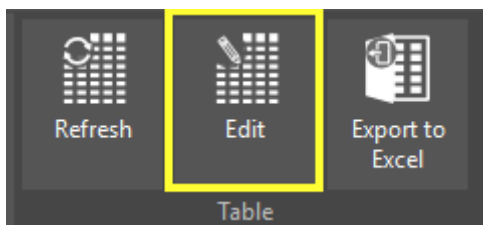


You should see a table looking like this one:

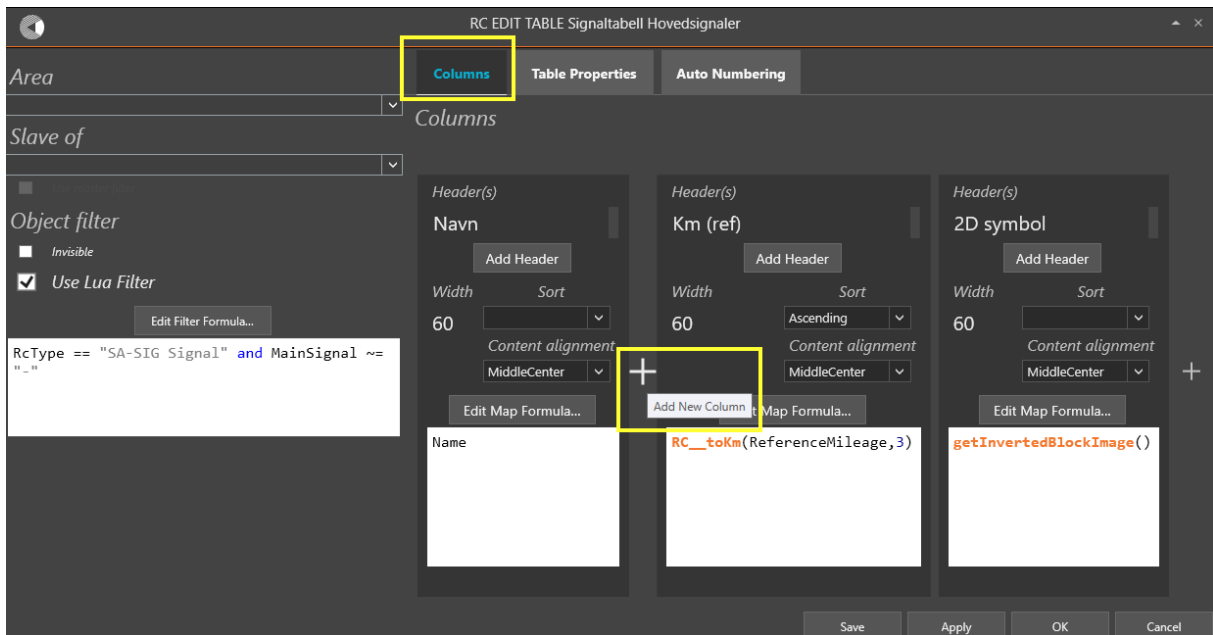
Signaltabell Hovedsignaler		
Navn	Km (ref)	2D symbol
Hs.12354	462.230	
Hs.12353	462.265	

Modify table

To modify the table, click on the table, and select Edit in the ribbon bar.



In this following example we will extend the table by adding a column showing the 'own alignment' and we will modify the existing 2D-symbol column to make the table contain 'photographs' of the objects' 3D symbols instead of showing the 2D symbols.



To add a new column, just hover the mouse pointer to a position besides or between existing columns in the table 'Columns' Tab, where the new column is supposed to be added. Click on the '+' sign to add the column. Change the header (column caption) from 'New Column' to 'Alignment', and change 'Name' into 'Alignment' in the white field below the 'Edit Map Formula...' Lua editor button.

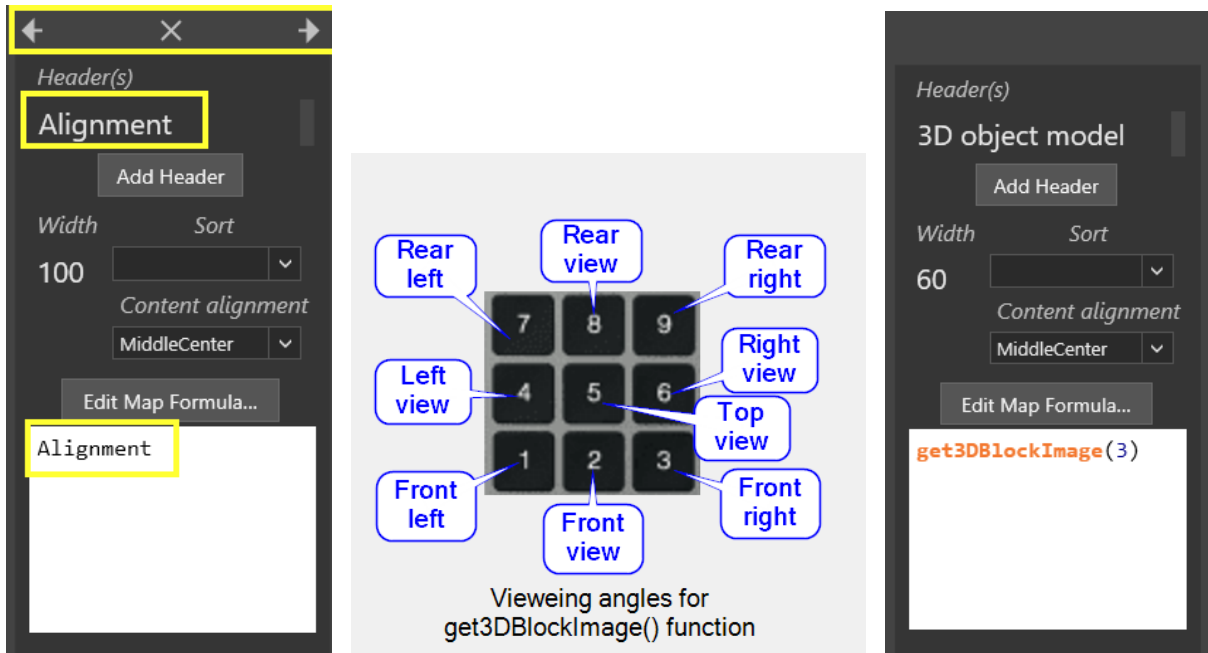
Please notice the two arrows and the cross that appear if you hover over the top of one column. Using the arrows, you can move the column to either side to change the appearance of your table. You may also drag-and-drop a column to its new place. Clicking the cross will delete the column. Be sure to use the Save button often, to write your table under development back to your drawing. A deleted column cannot be retrieved unless you have a saved copy of your table to go back to.

Pressing 'Save' will write the current table editor contents to the table object stored in your drawing.

Pressing 'Apply' will refresh your table without leaving the table editor tool.

Pressing 'OK' will close the table editor and apply the changes to your table.


Pressing 'Cancel' will terminate the table editor tool without writing to your drawing. The previously saved table version will be kept unchanged.



Of course, you may 'tamper' with the resulting CAD system table object directly, but these changes will be overwritten by RailCOMPLETE the next time you refresh (update) your table.

In the 2D symbol column, modify the header from '2D symbol' to '3D object model' and change the Lua formula from '`getInvertedBlockImage()`' to '`get3DBlockImage(3)`'. The '3' corresponds to making a virtual photograph of the object's 3D model from a front right viewing angle, from a suitably positive Z coordinate. Think of a numeric keypad as the observer's standpoint, with the object being at elevation 0 under the '5' in the middle of the numeric keypad and the observer holding one of the positions 1..9 (0 has no associated viewing angle).

Note that the resulting table in AutoCAD will still show the 2D symbol while an export to Excel will show the 3D symbol. Select the table and press the 'Export to Excel' button in the ribbon.

	A	B	C	D
1	Signaltabell Hovedsignaler			
2	Navn	Alignment	Km (ref)	3D object model
	Hs.12354	[CROSSING-IN-TO-OUT]	462.230	
3				

The modified CAD table should now look like this:

Signaltabell Hovedsignaler			
Navn	Alignment	Km (ref)	3D object model
Hs.12354	[CROSSING-IN-TO-OUT]	462.230	
Hs.12353	[CROSSING-IN-TO-OUT]	462.265	


Export to 3D

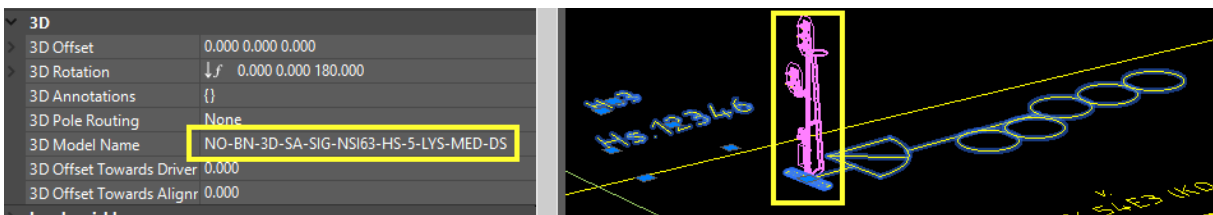
Typical workflow – 2D vs 3D

The typical workflow in RailCOMPLETE is that all modelling is done in 2D while comprehensive 3D visualization is added as an export to 3D.

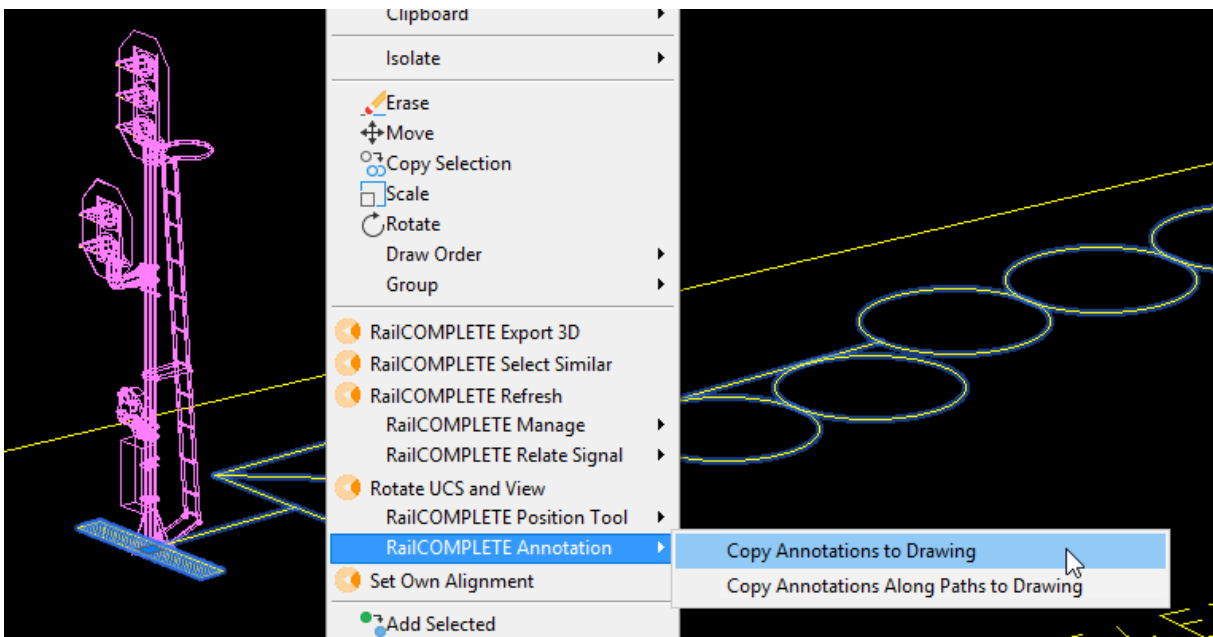
Separating 3D from 2D leaves your models much smaller in file size, which has a positive effect on file open/save waiting times, storage space, the ability to transmit models in an email to your peers etc. 3D object models for RailCOMPLETE should ideally be made only out of AutoCAD SOLID entities, because RailCOMPLETE uses a level Of Detail (LOD) algorithm which is based on volumetric computations. 3D object models which are based on surfaces and regions will typically be much slower to work with.

3D preview tool

You may at any moment turn on the 3D preview tool , a 3D cube icon to be found in the ribbon in the ‘Show’ tab. When selecting an object with the 3D preview tool active, the object’s 3D model will “pop up” at the correct coordinates and with the correct rotation in 3D space according to the object’s 3D settings, to be found and manipulated in the Manage Properties “3D” section.

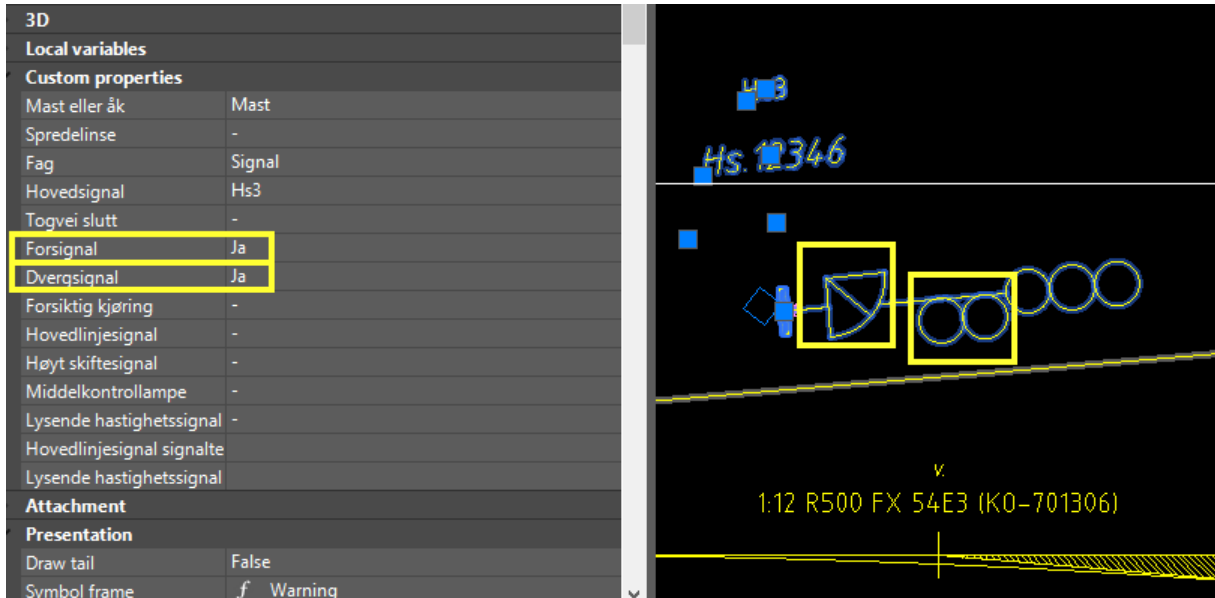


If you need to keep a “3D snapshot” in your model, e.g. to check free loading gauge etc, you can always copy the 3D preview to the drawing – it is treated just as any other annotation to the drawing.



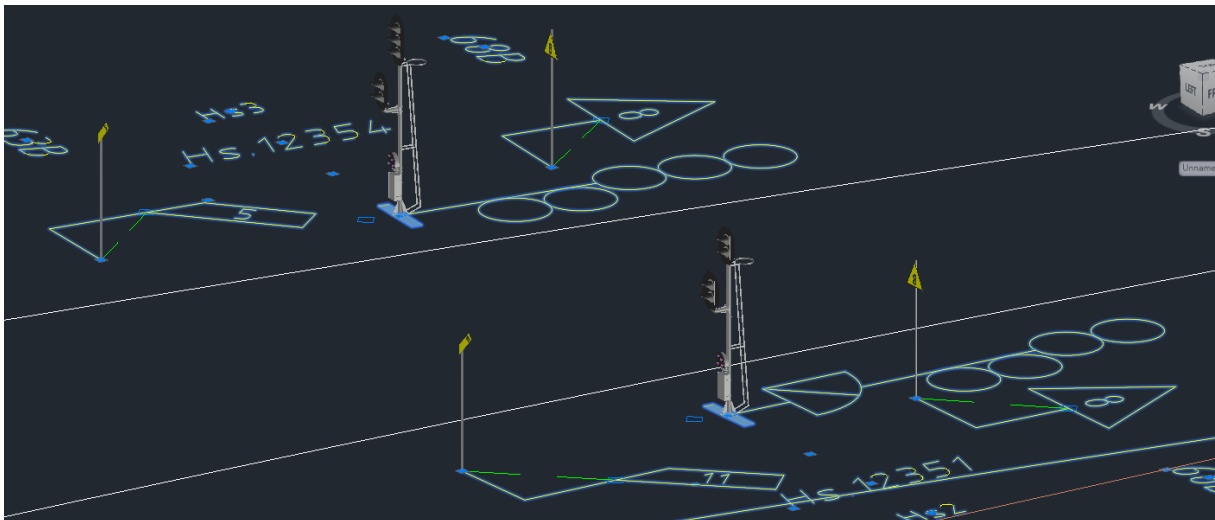
3D export example

In this example we will use two alignments, two signals and four boards. We changed the object properties of some signals from 3-lantern main signals into 4- or 5-lantern signals with dwarf signal, using the Manage Properties tool.

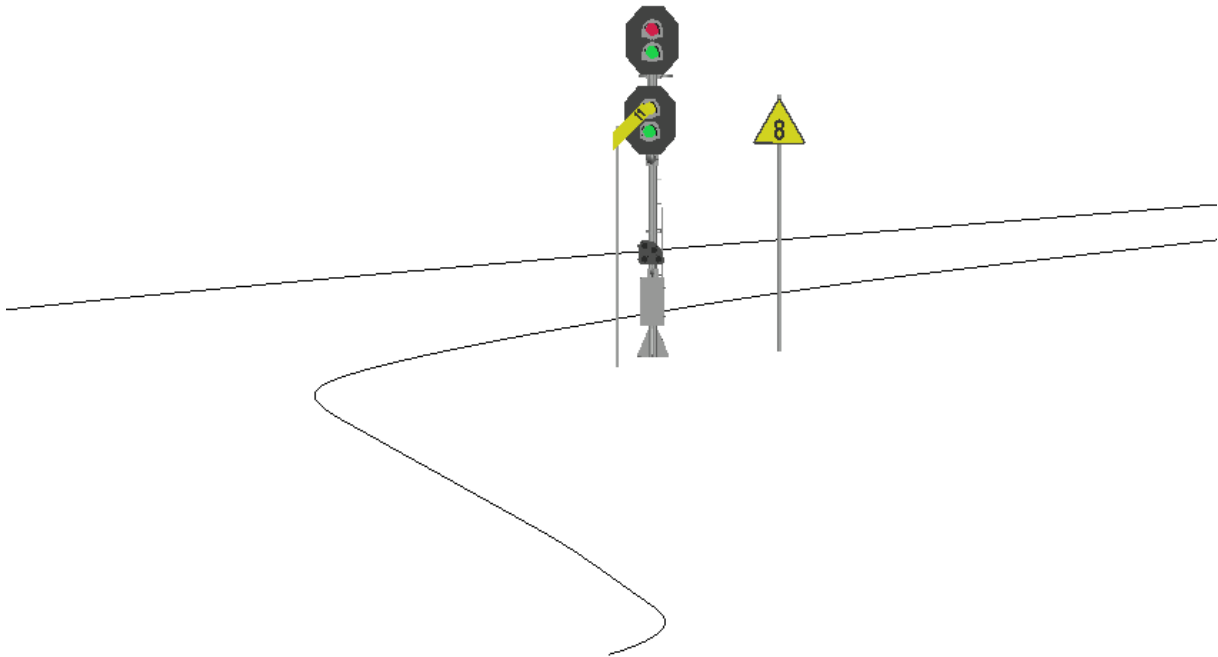


Boards were inserted using the RC-CreatePointObject command with object category "Skilt" (Norwegian for 'board') and inserting two of type '63B' (gradient boards) and two of type '68B' (speed boards).

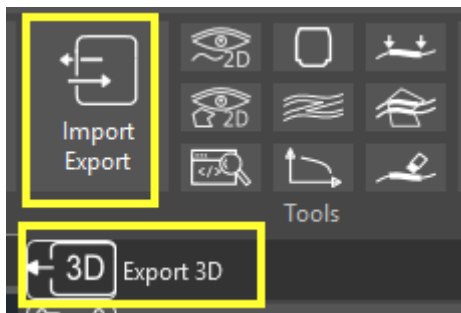
Here is what it looks like using the 3D preview tool:



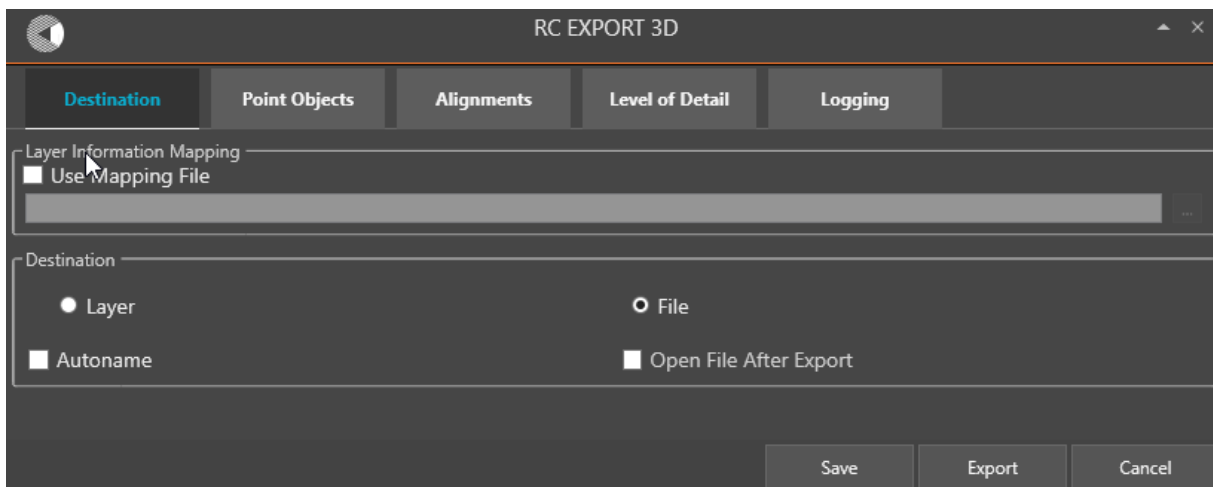
And here is what it should look like when using the 3D Export tool, exporting to a new file. AutoCAD 'Realistic' 3D view mode with white background was chosen. In the illustration below, we are focusing on signal sighting towards signal Hs.12351:

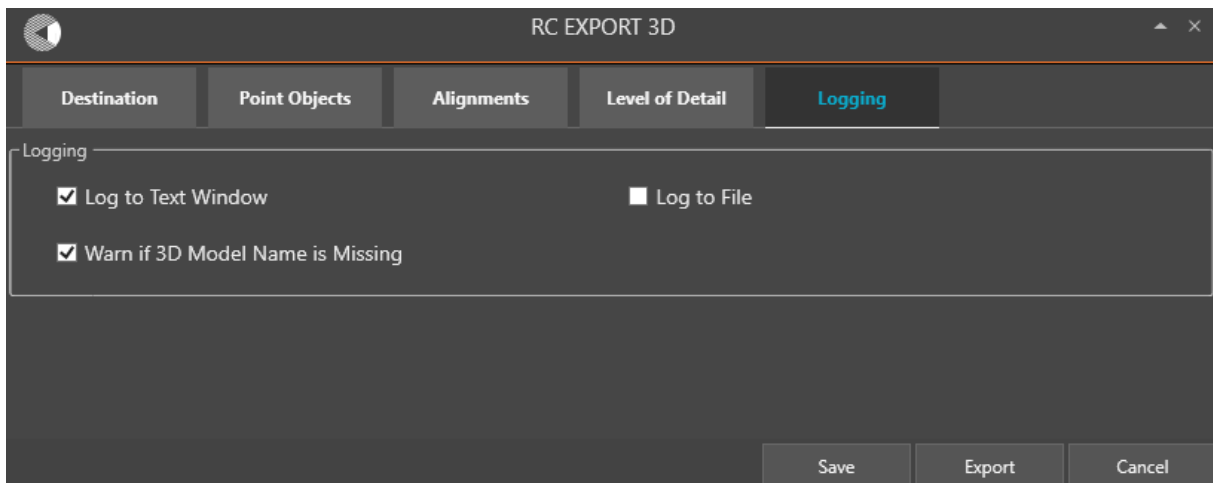
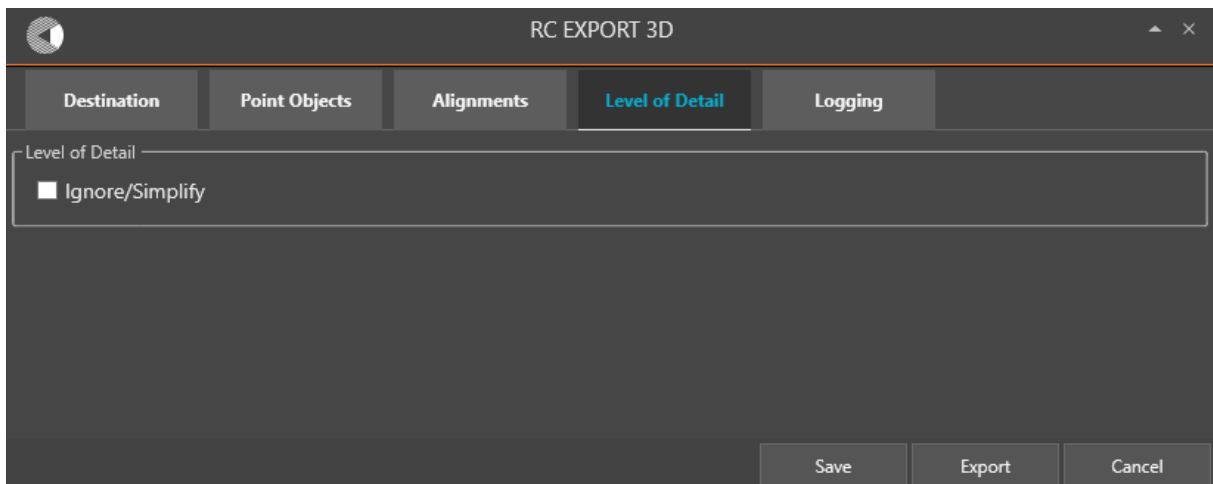
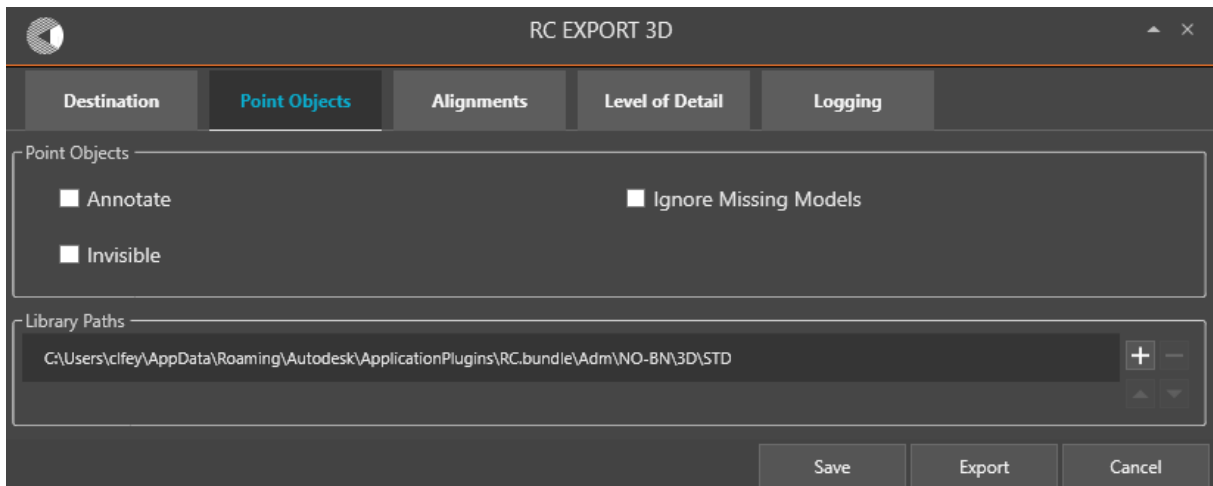


To use the 3D export tool, start it from the ribbon by selecting the Export 3D button below the Import/Export button, or launch the command by typing RC-Export3d.

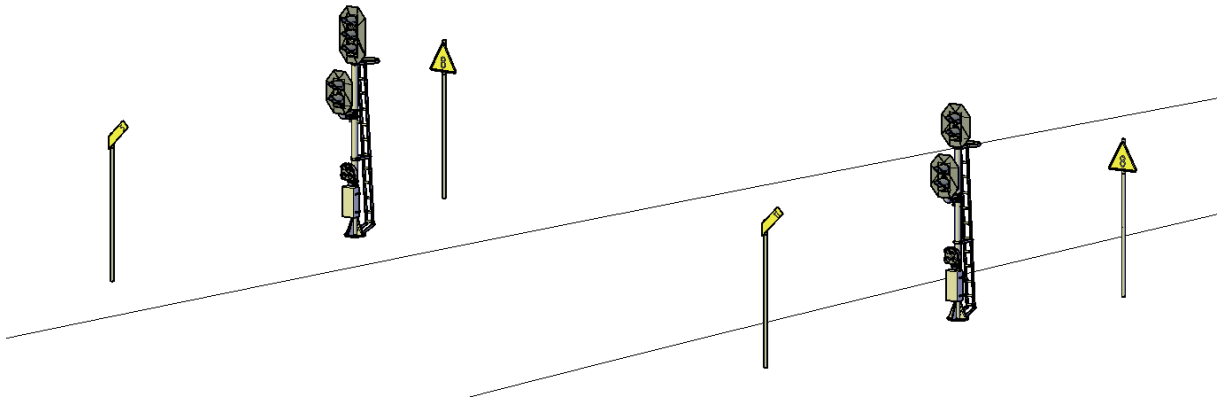


Use default tool settings as shown in the five illustrations below and then press the Save and then the Export button.





The result should look something like this.



In later versions of RailCOMPLETE, you may specify for each alignment object how it shall express itself in 3D. Earlier versions of RailCOMPLETE (as shown in the Alignment menu above) treated all alignments as if they were of the same type, i.e. you would have to modify the 3D export parameters for alignments each time another appearance was needed.

Try experimenting yourself with the various settings. There is support for multiple 3D object libraries, Level Of Detail control, textual annotation of both alignments and point objects, automatic file name generation, as well as a layer mapping tool (RC-ManageDrawingLibrary).

Object properties

Building Information Modeling (BIM)

RailCOMPLETE brings BIM (Building Information Modeling) to a whole new level. In addition to containing traditional BIM info – the manufacturer’s name, item ordering details, drawing archive reference, 3D appearance etc – RailCOMPLETE makes the objects smart and programmable through extensive use of the Lua scripting language in the object properties’ formulas.

RailCOMPLETE also features an efficient implementation of customizable binary relations linking objects to each other in a browsable web-like information network.

General data entry using the keyboard

In general, clicking in a cell or using the F2 function key brings you into the data entry mode. From there, Home/End, Up/Down, Left/Right, alone or together with the Alt, Shift and Ctrl keys, will have reasonable ways of working, similar to what you are used to from the Microsoft Office™ products.

The F3 key will always start the Lua editor.

The F1 key will generally provide context sensitive help information.

Modifying multiple objects simultaneously

Values that vary between the selected objects are displayed with ***VARIES***. As an example, to move a Modify multiple objects

Please select a group of similarly typed objects (e.g., RcType is “SA-SIG Signal”) and then open the Manage Properties window.

Values that vary between the selected objects are displayed with ***VARIES***. As an example, to move a group of selected signal objects by two meters, select the signal objects and add 2 to ***VARIES*** in the Mileage property row, then press ENTER.

Mileage	*VARIES*+2
---------	-------------------

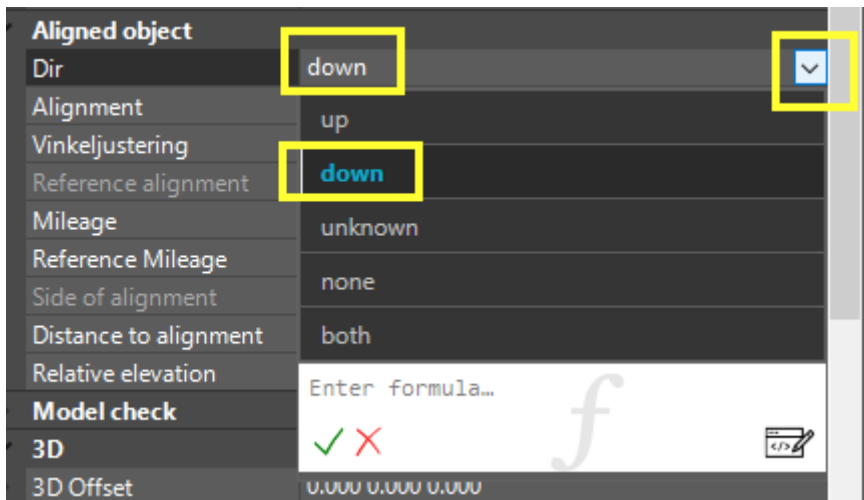
Data entry using the mouse scrollwheel

Enumeration values (predefined values that can be selected from a list, e.g. up/down/none/both, or object variants defined in the DNA) can be changed by double-clicking in the property’s value field in the Manage Properties tool. For each double-click, the enum advances by one step in the list of available values (‘down’ is followed by ‘unknown’ etc).

Alternatively, click on the down-arrow symbol to the right and select directly from the drop-down list.

With your cursor placed inside the property value field, acting on the mouse scrollwheel will bring you through the enum values in sequence, either up or down, depending on the scrolling direction.

The scrollwheel method does not work with enum property in multiple objects when they have differing enum values (***VARIES*** is shown). If the values are the same (their common enum value is displayed), then the scrollwheel method will work.



In later versions of RailCOMPLETE you can also use the mouse scrollwheel method as an alternative to typing in numbers in numerical fields. This method works also with multiple objects when *VARIES* is displayed, but only in properties that evaluate to a number.

Click in the property's value field (to the right) and move the mouse scrollwheel to increase or decrease values in steps of 1. Holding down the Control, Shift or Alt keys, or combinations of these, will alter the step size. See the table below for details.

Integer properties such as Seq (sequence) will only react to the integer step sizes 1, 10 and 100.

'-' means 'not held down'

'X' 'held down'

Ctrl	Shift	Alt	Numeric Action	Enum Action
X	-	X	+/- 100	No action
-	-	X	+/- 10	No action
-	-	-	+/- 1	Previous / Next enum
X	-	-	+/- 0.1	Previous / Next enum
X	X	-	+/- 0.01	No action
-	X	-	+/- 0.001	No action
-	X	X	+/- 1	No action
X	X	X	+/- 1	No action

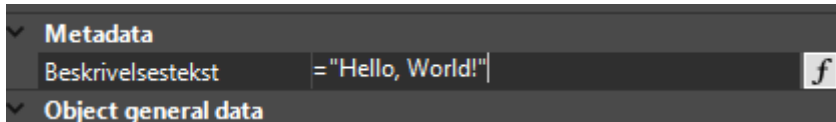
The combinations involving Shift+Alt are included for completeness but are not intended for regular use.

You may use the property manager and the mouse scrollwheel and the 3D preview tool together to manipulate your objects' 3D properties. This is a very powerful data entry method since you do not have to know the actual values at all, you only adjust them till you see with your own eyes that your items have been placed properly.

Lua programming

Lua is powerful and fast! Consult the RC-WebPagesAndFAQ command, it guides you to many good sources of information on Lua.

Most properties (data entry fields in the Manage Properties tool or in the Manage Objects tool) have the possibility to contain a formula. As with Excel, you may enter a formula directly in a cell by starting to type in an equality sign '='. Take for instance the 'description' property ('Beskrivelsestekst' in Norwegian), which holds a string value. Enter the formula ="Hello, World!" and press ENTER.

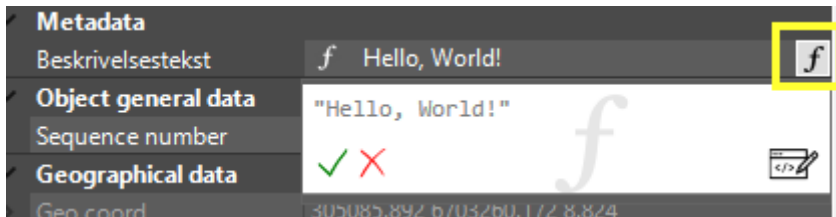


The formula evaluates to 'Hello, World!' and displays this text instead of the formula.

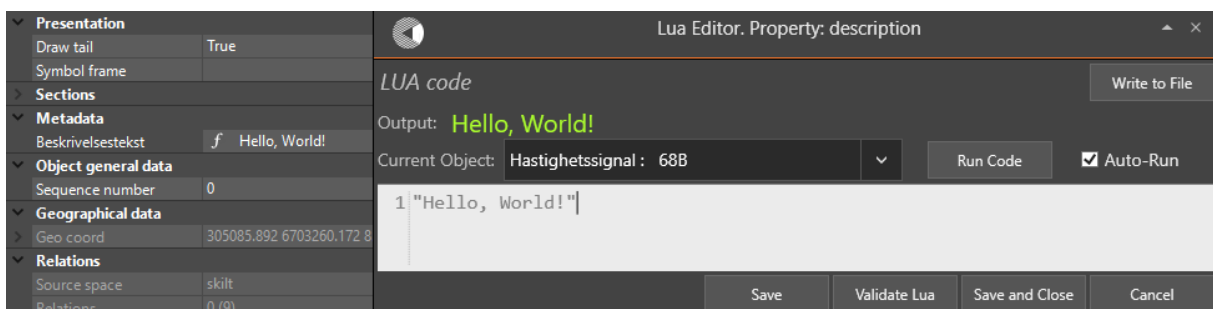


A little 'f' symbol appears, as a reminder that this cell now is driven by a formula. Hovering over the 'f' shows the contents of the formula.

To further edit an existing formula, just click in the cell and then click the rightmost 'f' symbol to bring up the small Lua editor:



Press F3 or click the "Edit" symbol in the lower right corner of the small Lua editor to start the larger and more powerful Lua editor:



The large Lua editor (usually referred as 'the Lua editor' or just 'the editor') allows you to use intellisense, i.e. auto-completion of reserved words. All predefined functions, from either the RailCOMPLETE kernel software or from the administration's DNA, may have a detailed description which shows up when you hover over a reserved identifier.

Lua example

In this example we are going to give automatic description for a signal. We want the output value to show 'Signal placed <Side of alignment> at mileage <mileage> on alignment <alignment>'.

Strings are embedded in simple (') or double (") hyphens. Concatenation of two text strings is expressed as string..string, i.e. using two periods between the two strings.

Comments start with a double dash (--) and extend to the end of line. Multiline comments use a double dash together with two opening / closing square brackets:

```
--[[ This is a...
...multiline...
comment. --]]
```

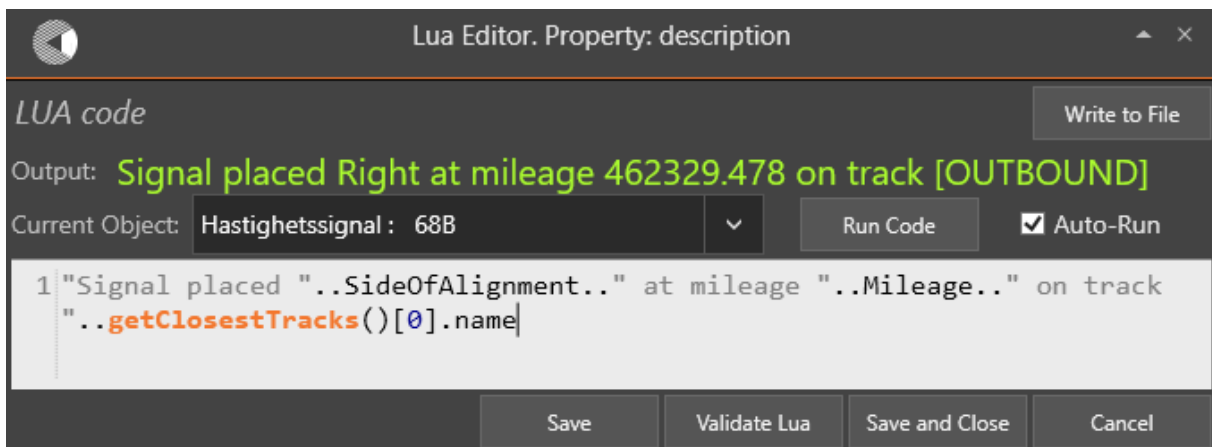
Delete the "Hello, World!" formula and enter this one instead:

```
"Signal placed "..SideOfAlignment.." at mileage "..Mileage.." on track "..getClosestTracks( )[0].name
```

Here, we have made use of the intrinsic properties 'SideOfAlignment' and 'Mileage'. We have also made a call to the intrinsic 'getClosestTracks()' routine which returns a collection – a list of railway tracks in the vicinity of the asking point object. This particular function also sorts the tracks in their order of closeness to the asking object.

To access items in this collection of tracks, we address them from 0 (closest) and up. If you need to find out how many items there are in a collection, use the getCollectionLength() function:

getCollectionLength(getClosestTracks()) will here return 6.⁴



The 'Output:' field will be updated as you enter the formula. If you make a typing mistake, then an error message will be shown.

Formula or Function?

Just as in Excel, we use the word **formula** to represent any valid Lua expression that can be evaluated 'as-is' by RailCOMPLETE to yield a value. A very simple formula would be an expression that returns the value '3.14'. In the Property Manager, just enter '=3.14' in an editable cell, for instance the property 'description'. This will now protect that cell's value from being inadvertently modified, since

⁴ Be careful with the method ".Count" that can be used as in 'getClosestTracks().Count', since items may be deleted from a Lua collection or Lua table but memory will not be freed up entirely and the call might return an incorrect count of 1. Use getCollectionLength(getClosestTracks()) instead, it always returns 0 if the collection has been emptied.

it reverts to 3.14 every time the object which is host for that property is evaluated. To modify a formula, bring up the Lua editor with F3.

The same value would be returned if you wrote just '3.14' in the cell, but typing any other value afterwards would change its contents, and you don't need the Lua editor to change the value.

A **function** is a declaration which includes a recognizable function name and which contains a Lua program (a sequence of Lua expressions), which can be called by name from another Lua expression, and which will return a result to the calling expression. Lua functions may take zero or more **arguments** and may return zero or more **return values**.

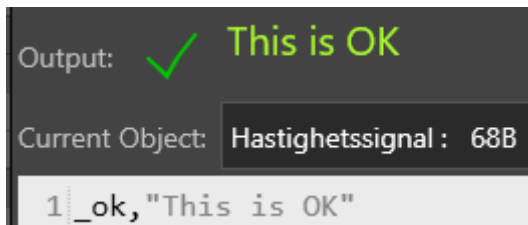
Return values in formulas


Formulas may return a single value or multiple values. A RailCOMPLETE property can only hold one number, enum, string value or object reference, but inside a Lua function you may use as many local or global variables that you need, each of them holding single or multiple values. A Lua function called by another Lua function may return several arguments. Just separate them with comma:


```
return a, b, c
```


RailCOMPLETE features reserved identifiers which we call 'symbols':

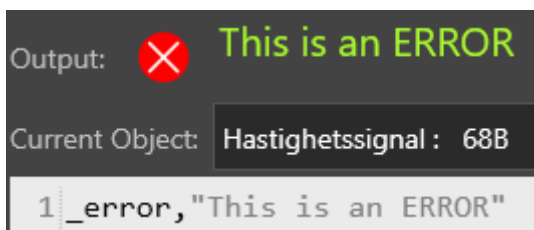
```
_noSymbol  
_ok  
_warning  
_error
```




Output:  This is OK
Current Object: Hastighetssignal : 68B
1 _ok, "This is OK"



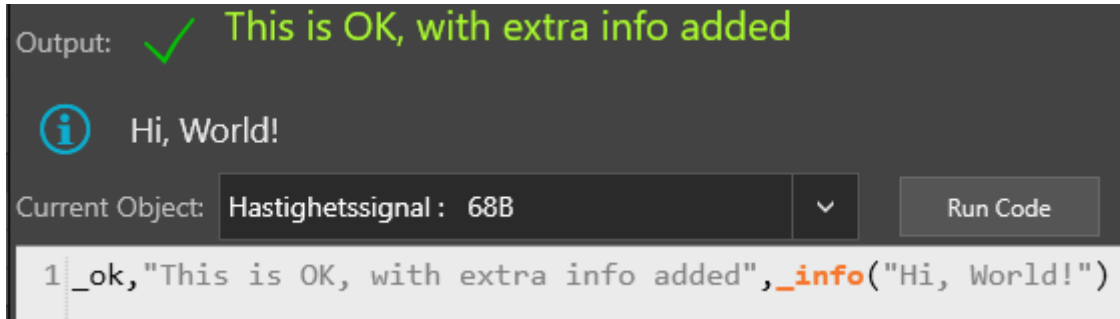
Output:  This is a warning
Current Object: Hastighetssignal : 68B
1 _warning, "This is a warning"



Output:  This is an ERROR
Current Object: Hastighetssignal : 68B
1 _error, "This is an ERROR"

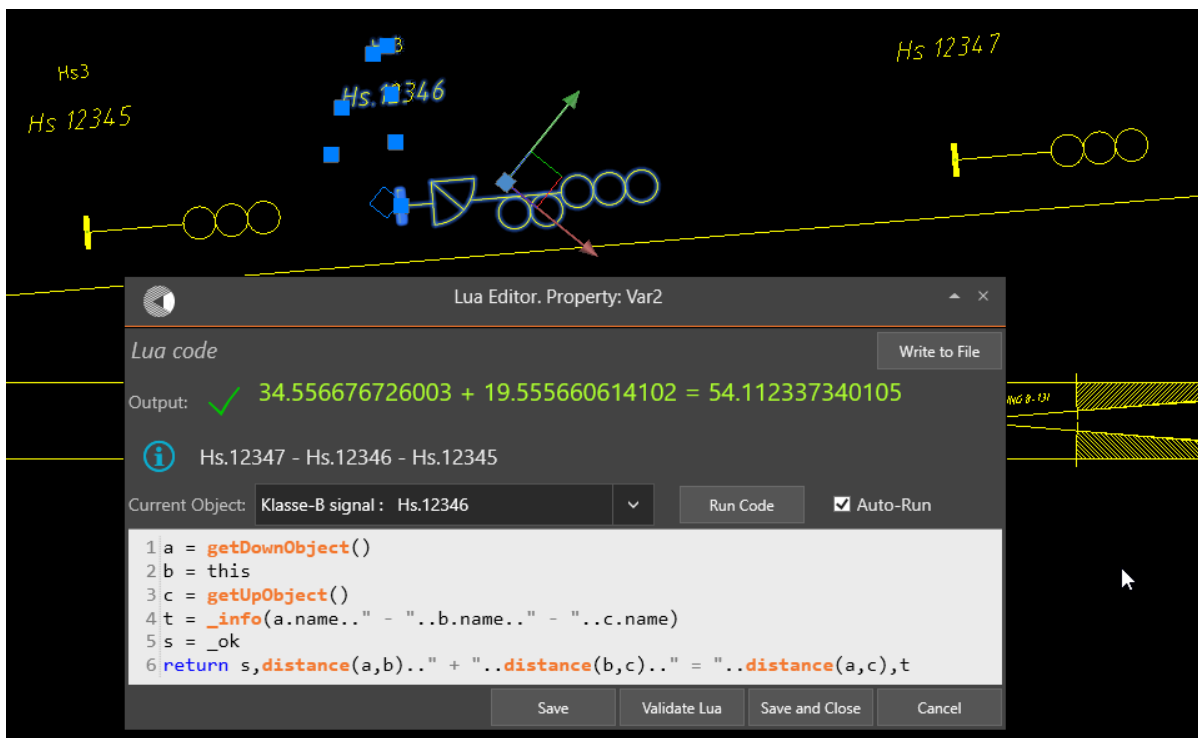
The `_noSymbol` is added for completeness, it generates no graphics in RailCOMPLETE.

Another special data construct that can be returned is the `_info()` item. It holds a string of text which is returned to the caller and displayed along a blue 'Info' icon, a small blue 'i' in a blue circle when presented in the property manager or in the Lua editor.



The order of appearance of the value, the symbol and the info item is of no importance. However, only the leftmost of each data type (value/symbol/info) will be used by the receiving RailCOMPLETE property.

Here is an example putting several concepts together:



Intrinsic RailCOMPLETE functions

Your custom Lua code may freely use identifiers which aren't already in use.

In the RailCOMPLETE kernel software there are many intrinsic functions that let you access the database in various ways. Generally, such functions are reading from the database, therefore their function name will often start with the letters 'get', as in 'getClosestTracks()'.

Below is a list of intrinsic RailCOMPLETE data access methods and reserved identifiers that may be used in your own Lua code. To see more details about each function, bring up the Lua editor, start typing in the identifier's name and hover your mouse over the identifier to see help info. Pressing F1

when hovering over an identifier (a word, a name) will bring you to the most relevant part of the Help system, which will usually contain useful examples along with syntax details and explanations.

- `distance()`
- `fromDisplayName()`
- `prop()`
- `getXxxxxYyyyyyZyyyy()` (many functions, notably `getPropertyValue()` and `getZeros()`)
- `isObjectReachableByPath()`
- `isVisible()`
- `orderByAscending()`
- `orderByDescending()`

Please consult the RailCOMPLETE reference manual for details on each function.

Native Lua functions

Lua comes with many native functions:

- `assert()`
- `error()`
- `filter()`
- `find()`
- `function` (reserved keyword)
- `ipairs()`
- `next()`
- `pairs()`
- `pcall()`
- `select()`
- `tonumber()`
- `tostring()`
- `type()`
- `unpack()`
- `xpcall()`

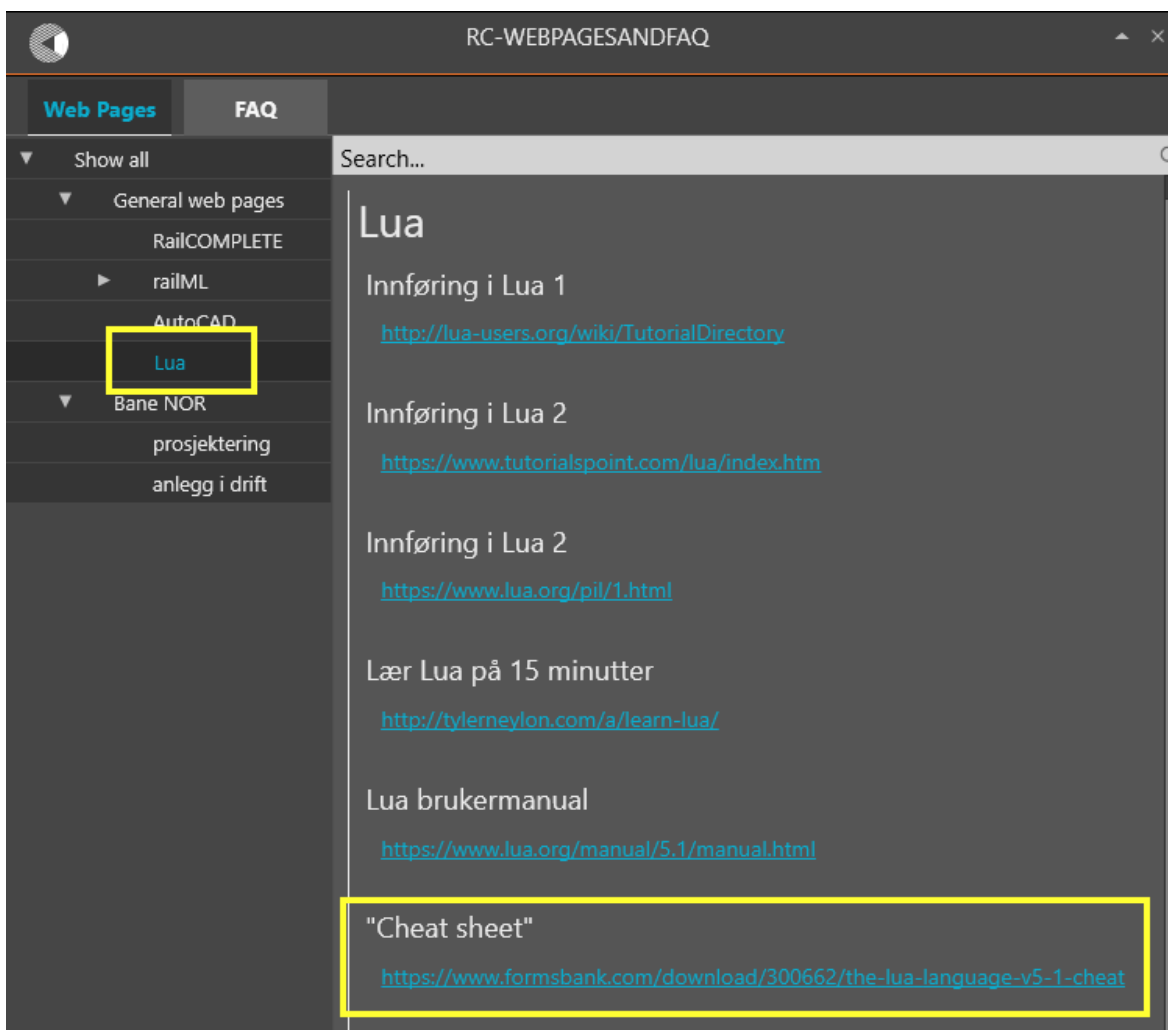
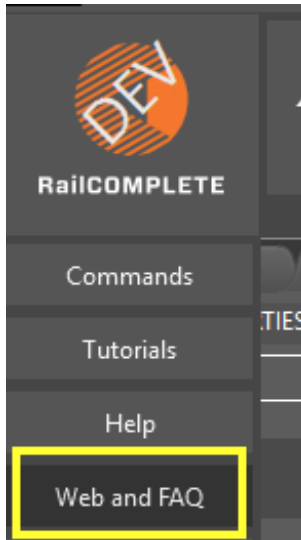
There are even more native Lua functions in the standard open source implementation, but they have been disabled by us for computer security purposes (file access and other operating system calls, notably).

We have included three important Lua classes: `math`, `string` and `table`. ‘`math.xxxx`’ denotes a mathematical constant such as `math.pi`, whereas ‘`math.yyyy()`’ is a mathematical function such as `math.sqrt(2)` or `math.cos(3.1416/4)`. The `string` class contains functions such as `string.upper()`, `string.lower()`, `string.match()` etc. The `table` class contains functions such as `table.insert()`, `table.sort()` etc.

A function (method) that has been declared in a class can be called even more elegantly by appending a colon character ‘`:`’ between the calling object and the method name, such as in

```
s = "ABCdef"
t = s:match("Cd")
return s:upper( ) -- an uppercased string
```


Please consult the information on Lua which you can find from the RC-WebPagesAndFAQ command, to get ideas and to become a better Lua programmer. You can find this command under the RailCOMPLETE logo.



What if a property name is the same as a reserved Lua keyword or identifier?

It is unavoidable that users from time to time come up with custom property names that are already in use as a reserved Lua keyword or identifier.

As an example, consider the railway markup language railML™, where wayside signals can be used for several purposes – such as ‘blocking’, ‘home’, ‘exit’ and ‘intermediate’. railML uses the property name ‘function’ to hold one of those four enumeration values. But ‘function’ is also a reserved Lua keyword, used to declare a custom function.

RailCOMPLETE solves this name clash by offering the intrinsic RailCOMPLETE function ‘fromPropertyName()’. To declare a Lua function that checks whether a signal object’s railML property called ‘function’ has the value ‘home’, write your code like this:

```
function isHomeSignal( )
    return fromPropertyName("function") == 'home'
end
```

Custom Lua function declarations in the DNA

In the RailCOMPLETE DNA for a specific railway administration, many functions have been developed by the RailCOMPLETE agent. These carry out functions which are specific to this administration, i.e. functions which are not universal and for that reason have not been included in the RailCOMPLETE kernel software.

Below is an example of a function that sets a basic tolerance scaling factor, for use as a constant in automated checking of models. The idea here is that all subsequent checking functions will use calls such as ‘tol = 0.300*RC__EPSILON()’ instead of hard-coding the tolerance to ‘tol = 0.300’ in each model checking function. This is what a function declaration can look like in the DNA:

```
<LuaFunction Name="RC__EPSILON()" ReturnType="Double"
  Description="Constant: Returns epsilon, a tolerance scaling number, to be used in model checks." >
  <Constructor>Double RC__EPSILON() </Constructor>
  <Formula>
function RC__EPSILON()
  return 1.000
end
  </Formula>
</LuaFunction>
```

A function may be declared inside a specific object type declaration in the DNA, making this function accessible only to objects of that type (e.g. a signal, a cantilever, a mast). A function may also be declared globally in the DNA and will be accessible from anywhere in a RailCOMPLETE model.

Lua function naming

Lua functions declared in the DNA should have names with a recognizable syntax, to make life easier when sharing Lua code between DNAs from different administrations.

By convention, general Lua function names should start with the letters ‘RC’ (for ‘RailCOMPLETE’). These functions are specific to the railway business but not to any particular administration – or they are just not included yet in the pool of intrinsic RailCOMPLETE functions.

Examples

```
RC__EPSILON( )
RC__getDistance2D( )
```

```

RC__isNan( )
RC__round( )
RC__setSymbolFrame( )
RC__snap( )
RC__toKm( )
RC_com_getLabelItem1( )
RC_com_getNearestEntireKm( )

```

By the same convention, DNA-declared functions for one specific railway administration should have names starting with the appropriate 2-letter ISO 3166-2 country code, immediately followed by a short letter combination representing the railway administration in question. For instance, Norway / Bane NOR may have 'NOBN_xxxxYyyyZzzzz()' function names, whereas the US railway company Burlington National may have USBN_xxxxYyyyZzzzz(). France / SNCF Réseau may have 'FRSR_xxxxYyyyZzzzz()' functions, and Germany / Deutsche Bahn may have 'DEDB_xxxxYyyyZzzzz()', and so on.

Examples

```

NOBN_com_chkNumberOfOcpAreas( )
NOBN_bnp_getBoardSightingRequirement( )
NOBN_trk_getLiftFromCant( )
NOBN_sig_getSignalShortName( )
NOBN_ocs_getCantileverModel3dName( )

```

The rationale behind this unique naming of such functions is to ease the process of “Darwinism”, i.e. people publishing a function they are satisfied with, and others casting it into their own administration’s needs but under a different name. We should collectively avoid that two functions ever exist with the same name but with different purposes or interpretations.

Lua function naming conventions in RailCOMPLETE

General and administration-specific Lua function names should:

start with the appropriate letter combination	X [X = RC NOBN FRSR DEDB etc]
followed by one underscore	X_
followed by a railway technical discipline identifier	X_com [void is allowed]
followed by one underscore	X_com_
followed by either a capital-letter constant name	X_com_MYCONSTANT()
or a camelCased function name	X_com_myFirstFunction()

If there is no suitable discipline name associated with the function, then this part is voided and you will see two underscores after one another, as in 'RC__round()'.

By convention, the railway electrotechnical discipline abbreviations might be:

```
(void) / com / sub / trk / cur / ocs / sig / pow / tel / bnp / otr
```

with the following meanings:

```

(void)  General non-railway-specific issues, such as rounding numbers, format conversions etc
com     Common - general railway-specific issues
bnp     Boards and ('n) poles

```

sub	Substructure (civil engineering, foundations, roads, buildings, bridges, tunnels, platforms etc)
trk	Track and ballast, switches, crossings, buffer stops
cur	Traction current system, transformers, power distribution system
ocs	Overhead catenary system
sig	Signaling
pow	Power supply, except traction current system
tel	Telecommunications
otr	Other systems

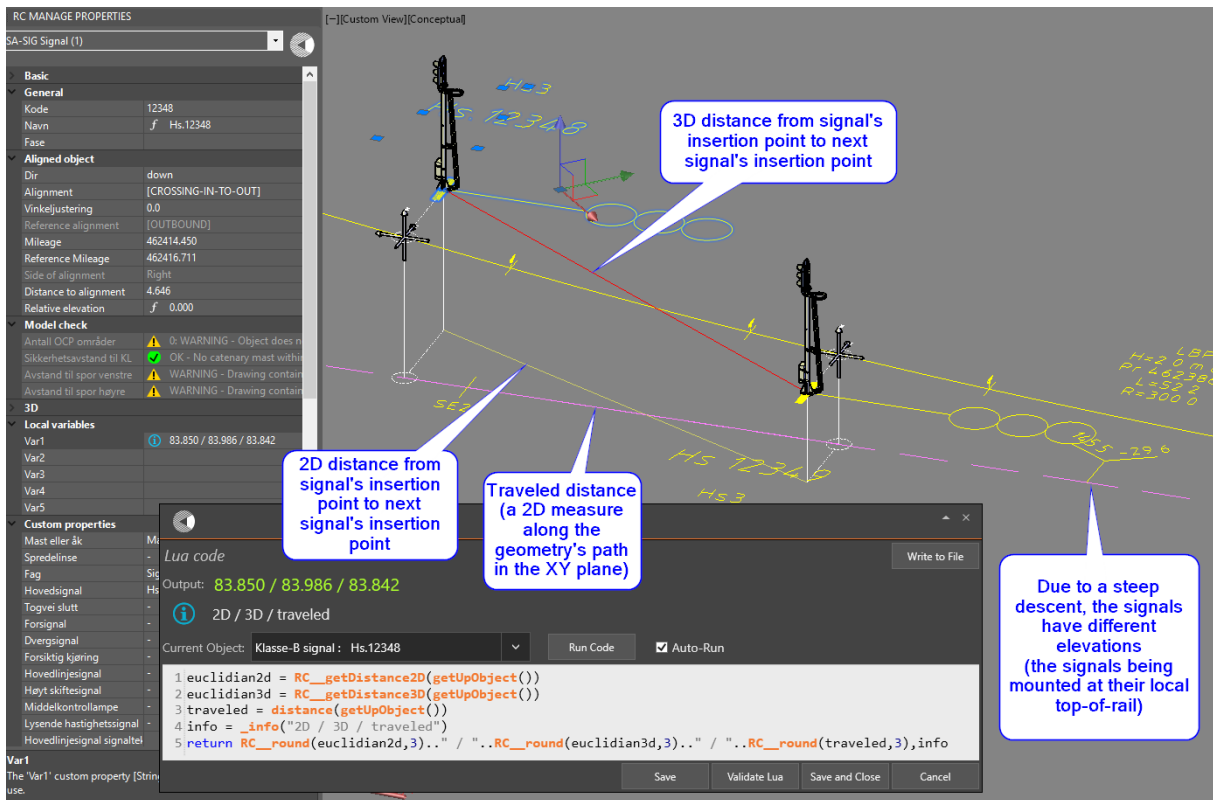
Also by convention, the identifying name of the function should be structured as an **action** followed by an **object**, such as in 'getData', 'setValue', 'chkValuesAgainstTolerances' etc.

get	Functions that read data from the database or acquire a result somehow, should have an identifying camelCased name starting with 'get'.
set	Functions that perform a calculation to set a property to a certain value should have an identifying camelCased name starting with 'set'.
chk	Functions that perform a sanity check or check other values against known tolerances should have an identifying camelCased name starting with 'chk'.

In summary: **<PREFIX>_<category>_<actionObjectName | CONSTANTNAME>([arglist])**

The example below illustrates the use of the intrinsic distance() and getUpIbject() functions, as well as the custom RC__round(), RC__getDistance2D() and RC__getDistance3D() functions.

Note that the Euclidian 3D (straight-line in space) is always longer or equal to the Euclidian 2D distance (ignoring the Z coordinate). However, the traveled distance along the object's path (alignment centre line) can be either shorter than, equal to or longer than the Euclidian 2D distance, depending on how far from the track the objects are placed, and how much the track is winding between the objects.



Constants

Constant names should be stated without an action, just an uppercased noun part, with underscore as word separator.

Constants' identifying names shall should be written in uppercase with underscore as word separator.

```
NOBN_trk_STD_GAUGE( )
NOBN_sig_'BRAKINGCURVE_TARGET_DISTANCE( )
```

Constants can be single values or tables which can used as-is or further indexed.

```
RC__EPSILON( )           returns a general constant 'epsilon'
NOBN_trk_SINGLE_VALUE ( ) returns a single value '1.5'
NOBN_trk_2DTABLE( )     returns a table {{3,5,8,13},{5,8,13,21}}
NOBN_trk_2DTABLE( ) [2][4] returns '21' from {{3,5,8,13},{5,8,13,21}}
```

Functions expect a variable number of arguments and return a list of arguments, usually only one, or one argument and a symbol (to be shown in modelchecks or in tables along with its first return value).

Arguments

Lua functions may take any number of arguments.

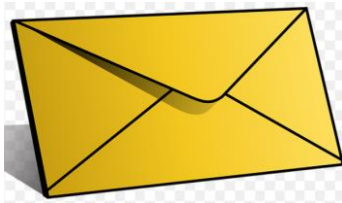
For instance, function xyzzy(a,b,c) could be declared in the DNA with three arguments named a, b and c, but the caller might pass on only two, one or no arguments instead of all three.

- If function xyzzy is called as xyzzy(1,2,3) then a=1, b=2, c=3 would be passed on to a, b and c.
- If called as xyzzy(2,3) then we would have a=2, b=3 and c=nil.
- Calling xyzzy() with no arguments would lead to a=b=c=nil.

Programming style – Way Of Working (WOW!)

We urge every Lua code writer to respect the following general programming rules for Lua functions:

- Test the arguments for being *nil* and take appropriate action if one or more are not present
- Check the consistency of arguments with other knowledge available
- Be systematic and define an action for all possible inputs
- Always ‘code for the impossible’ – i.e. test for all possible input values, but also include a default clause that catches situations that are ‘known to be impossible’, such as enums not being any of the allowed values. Your function might be newer than the data you are applying it to, the DNA might have been updated with new or modified enum names etc – be cautious.
- Use standardized indentation, casing and parenthesis settings to increase code readability
- Use ample space (empty lines) between functions
- Comment your code
- If applicable, return a symbol (`_ok`, `_warning`, `_error`)
- If applicable, return an info message (`_info(<text>)`)
- Subject your code to peer reviews, let your colleagues test – and read – your code

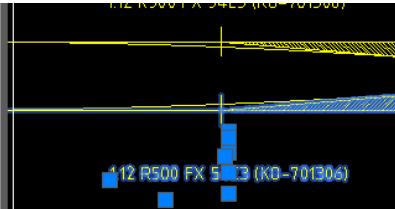


Please post your code on the forum pages or send your proud suggestions to support@railcomplete.com.

Conventions for Model Check functions

RailCOMPLETE objects will typically contain properties which are tagged as ‘ModelChecks’. Such properties can not be “tampered” with by the user, they are already locked in their ‘genetic code’ as declared in the drawing’s DNA. The purpose of such properties is to make a sanity check based on the object’s own data and surrounding or related objects’ data, and to flash a warning or an error if problems are found. It is customary to use the objects’ standard mechanism for adding a so-called ‘symbol frame’ around the 2D graphics in order to show one framing for warnings and another for errors.

Model check		
Antall OCP områder	✓	XYZ: OK - Object belongs to just one OCP area.
Alignment type	✓	KO-SPO Spor/KO-SPO Spor: OK - Alignment types are compatible.
Continuity	✓	0.000: OK - Alignments meet at connection within 50% of tolerance 0.030 m.
Tangent direction	✓	-0.000: OK - Tangents match within 50% of tolerance +/- 0.300 degrees.
Elevation	✓	-0.000: OK - Elevations match within 50% of tolerance +/- 0.010 m.
Gradient	✓	0.000: OK - Gradients match within 50% of tolerance +/- 0.300 o/oo.
Cant	✓	0.000: OK - Cants match within 50% of tolerance +/- 0.010 m.



Model check functions should return a string:
Starting with the assessed value...

followed by a colon...

followed by a space and one of the words 'OK' / 'WARNING' / 'ERROR'...

followed by space - dash - space (' - ')...

followed by a full sentence starting with capital letter and ended by a period '!'...

followed by one of the predefined symbols in RailCOMPLETE: `_error` / `_warning` / `_ok`

A call to `RC_mc_NumberIsSmallEnough()` with assessed value `n == 0`, `1` or `>=2` will then return these:

"0: OK - Number is small.", `_ok`

"1: WARNING - Number is still acceptable but close to the limit.", `_warning`

n..": ERROR - Number is unbearably large", `_error`

A suggested practice is to let the warning threshold be half of the error threshold.

Variables, functions and their scope

Most functions In the RailCOMPLETE DNA there are many functions being declared as global, i.e. they can be reached from any object's property.

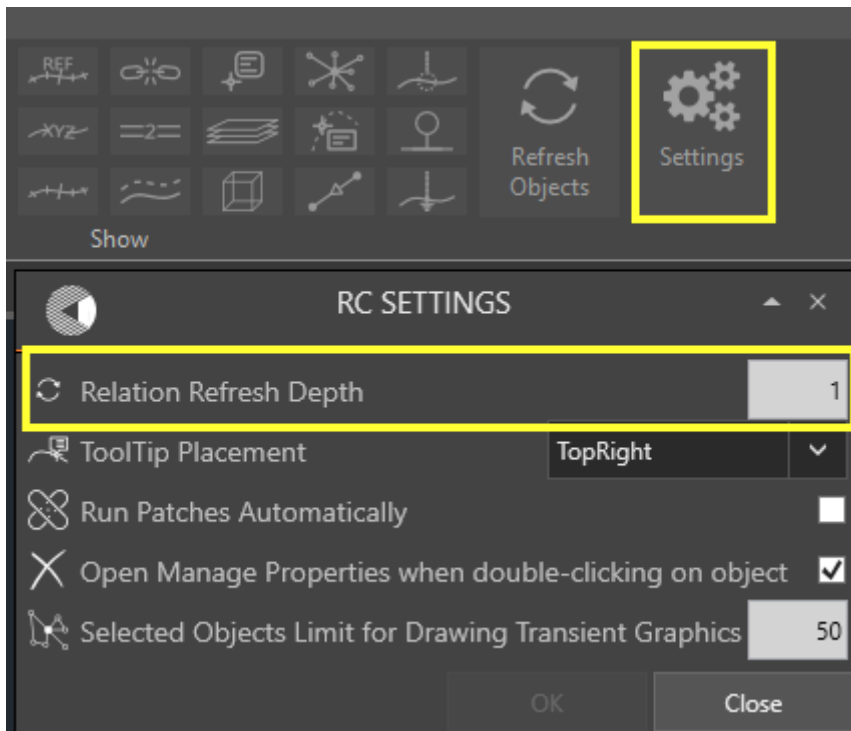
Refreshing objects or not

The act of reading an object's property or properties will usually trigger a 'refresh' of that object before it returns any values to you or to your Lua functions. This ensures that formulas have been evaluated before their values are used.

If a loop is detected where one object depends on another, which directly or indirectly depends on the first object, then an error message is shown and execution stops.

There is, however, a way to avoid such loops. You may use the intrinsic function `getPropertyValue()` to read a property without evaluating its formula first.

Objects may be interrelated using so-called '[binary] relations' (see a later chapter in this tutorial). One object can reach lots of other objects through such relations, and those objects can again reach lots of new objects. All such reachable objects should ideally be evaluated before your object uses property values contained in those other objects. This would however lead to an explosion of computations, so RailCOMPLETE sets an upper limit to how deep relations are followed in the refresh process. A relation depth of 1 or 2 is generally OK. If you have a very heavy model, then you might consider setting the depth to 1 temporarily until all data have been entered, before you start producing tables and etc.




Copy formula

In this example we will copy the formula to all other signals using the object manager.

Open the object manager, use filter to get all signals. Make sure the Description column is displayed and select all target cells as well as the source cell.





Select Copy Formula

type	Description	Na
-SIG Signal	<i>f</i> Signal plac	<i>f</i>
-SIG Signal		<i>f</i>
-SIG Signal		<i>f</i>
-SIG Signal		<i>f</i>
-SIG Signal		<i>f</i>
-SIG Signal		<i>f</i>
-SIG Signal		<i>f</i>
-SIG Signal		<i>f</i>

- Select
-  Zoom
- Select and Zoom

- Find and Replace

- f* Set formula
- x**f* Replace formula with current value

-  Copy values and formulas (CTRL-C)
-  Copy special ▶
-  Paste (CTRL-V)
-  Paste special ▶

- Auto-fill: Copy Cells
- f* Auto-fill: Copy Formula
- Auto-fill: Fill Series

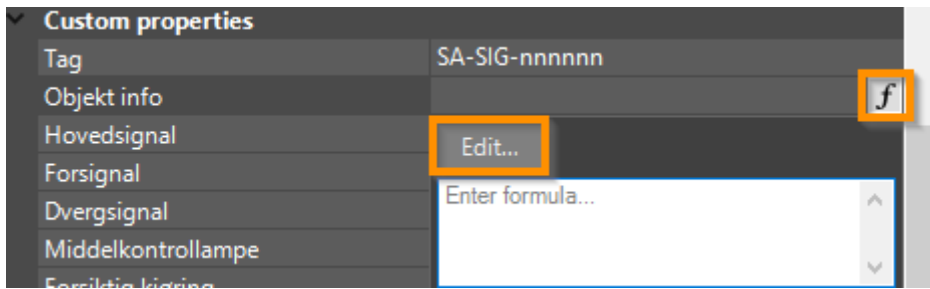
The result should be something like this:

Description
<i>f</i> Signal placed Right at mileage 949.822 on track CROSSING 8-13
<i>f</i> Signal placed Left at mileage 847.822 on track INBOUND
<i>f</i> Signal placed Left at mileage 1428.0590780446 on track CROSSING-IN-
<i>f</i> Signal placed Right at mileage 1430 on track OUTBOUND
<i>f</i> Signal placed Right at mileage 549.822 on track INBOUND
<i>f</i> Signal placed Right at mileage 647.822 on track INBOUND
<i>f</i> Signal placed Right at mileage 749.822 on track TURNAROUND
<i>f</i> Signal placed Right at mileage 849.822 on track TURNAROUND

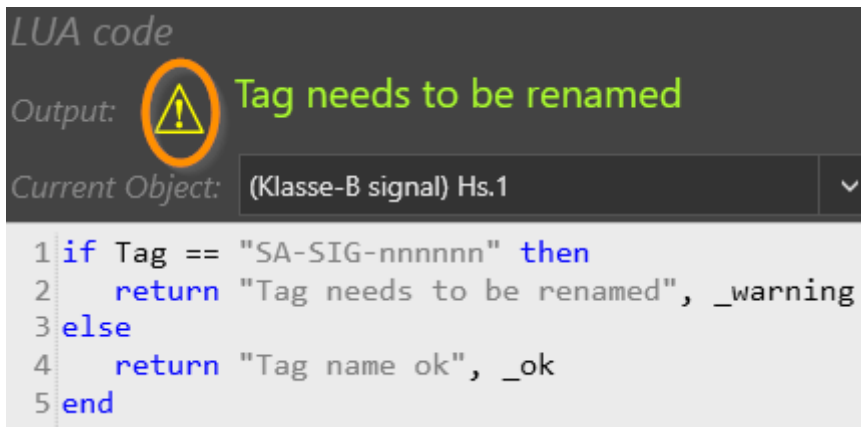
A simple Lua program

In this example we are going to give a warning if the signal tag still has the initial value of “SA-SIG-nnnnnn”.

Select the Object info field in the properties window and open the Lua editor.

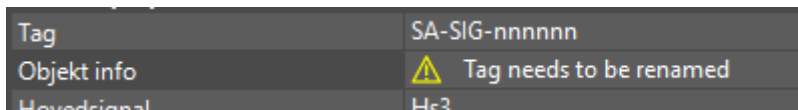


The program to check for tag value is strait forward. Enter the code as shown below.



Note that the `_warning` and `_ok` at the end of the return statement also gives a warning and valid-sign.

Press OK, and check the Object info.



Now go and modify the Tag field to i.e. SA-SIG-001234.



If we go into the object manager and add Tag and Object info, we get the same.

RC type	Alignment	Objekt info	Tag
iA-SIG Signal	OUTBOUND		SA-SIG-nnnnnn
iA-SIG Signal	INBOUND		SA-SIG-nnnnnn
iA-SIG Signal	INBOUND		SA-SIG-nnnnnn
iA-SIG Signal	INBOUND		SA-SIG-nnnnnn
iA-SIG Signal	INBOUND		SA-SIG-nnnnnn
iA-SIG Signal	INBOUND		SA-SIG-nnnnnn
iA-SIG Signal	CROSSING-IN-TO-OUT-0		SA-SIG-nnnnnn
iA-SIG Signal	INBOUND	✓ Tag name ok	SA-SIG-0001234
iA-SIG Signal	INBOUND		SA-SIG-nnnnnn
iA-SIG Signal	INBOUND		SA-SIG-nnnnnn

If we copy the program in Object info to all other signals, the object manager will show these values.

Objekt info	Tag	Side of alignment
✓ Tag name ok	SA-SIG-0001234	Left
⚠ Tag needs to be rename	SA-SIG-nnnnnn	Right
⚠ Tag needs to be rename	SA-SIG-nnnnnn	Right
⚠ Tag needs to be rename	SA-SIG-nnnnnn	Right
⚠ Tag needs to be rename	SA-SIG-nnnnnn	Right
⚠ Tag needs to be rename	SA-SIG-nnnnnn	Right
⚠ Tag needs to be rename	SA-SIG-nnnnnn	Right
-0	SA-SIG-nnnnnn	Left
⚠ Tag needs to be rename	SA-SIG-nnnnnn	Left

Using the object manager to update the Tag values, will also update the Object info at the same time.

Objekt info ▼	Tag	Side of alignment
✓ Tag name ok	SA-SIG-0001234	Left
⚠ Tag needs to be rename	SA-SIG-nnnnnn	Right
✓ Tag name ok	SA-SIG-123456	Right
⚠ Tag needs to be rename	SA-SIG-nnnnnn	Right
⚠ Tag needs to be rename	SA-SIG-nnnnnn	Right
⚠ Tag needs to be rename	SA-SIG-nnnnnn	Right
⚠ Tag needs to be rename	SA-SIG-nnnnnn	Right
⚠ Tag needs to be rename	SA-SIG-nnnnnn	Left
⚠ Tag needs to be rename	SA-SIG-nnnnnn	Left
⚠ Tag needs to be rename	SA-SIG-nnnnnn	Left

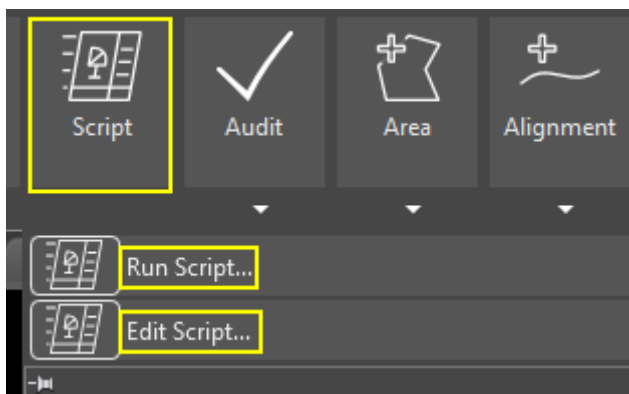
Scripts in Lua

A RailCOMPLETE **script** is a stand-alone program written in Lua, intended for use inside RailCOMPLETE. It is stored on your file system, and not stored inside your CAD system file where your objects (with formulas) reside.

RC-RunScript and RC-EditScript

The intention with scripts is to let you make a general program that can be applied to many CAD system files, for instance a program to purge unused blocks, or produce a 3D export to another file. A script might as well read and interpret object database files from real operations and from survey data and rapidly build a 3D BIM model, serving as a basis for further engineering designs.

The RailCOMPLETE scripting tool runs a stand-alone Lua program. You will find two commands in the ribbon, the **RC-RunScript** and the **RC-EditScript** commands. The former will ask for a Lua file name and then execute it. The latter opens an editor where you can edit multiple files as well as start / stop your script and have access to the logging as it runs. Although RC-EditScript is not a full-blown Lua debugger, it features Lua syntax validation, stack trace and runtime error messages.



Sandboxing

A Lua expression which is contained in or called from a property belonging to a RailCOMPLETE object is not allowed to access your file system in any way and has no access to the keyboard or other user input as it evaluates. This is known as a **sandbox** – where you can safely play without compromising the world outside.

A Lua script is not sandboxed. It can read and write files and can prompt the user for input as it runs, for instance selecting files or folders.

The Lua scripting API

The RailCOMPLETE API – Application Programming Interface – is a collection of functions offered by the RC-RunScript and RC-EditScript commands to enable Lua expressions to get and set data.

Important API functions are:

`runCommand(<a command to be executed by the CAD system, including RC commands>)`

`askForFile(<prompt text shown to user when a file reference is to be returned>)`

`getContentsFromFile(<reference to a file or a folder, as well as its filetype>)`

writeContentsToFile(<reference to a file or a folder, as well as its filetype>)
askForObject(<message string>)
createPointObject(<data required to define a new RailCOMPLETE point object>)
movePointObject(<object reference and its new coordinates>)
setSelectionSet(<references to RC- and non-RC-objects which will form the selection set>)
setRelation(<create a named binary relation between a source and a target object>)
write(<text string to be outoutput to the script's console window>)

A typical example calling on one of the RailCOMPLETE commands:

```
setSelectionSet( { obj1, obj2, obj3 } ) -- select three RC objects...  
runCommand("RC-Export3dUsingCurrentSettings") -- ...then export to 3D
```

A typical example using the host CAD system to draw a circle with radius 5 around origo:

```
runCommand('._CIRCLE "0,0" 5.0 ')
```

Note that the CAD system command needs to put quotes around some arguments (such as the coordinates "0,0"), and so does the Lua runCommand() as well. This conflict is easily solved because Lua recognizes both single quotes (') and double quotes ("). One type may enclose the other, Lua does the quote matching by itself. By convention, use single quotes for Lua and double quotes for the CAD system.

Also note the presence of a single space ' ' character at the end of the CAD command string. This space character plays the role of the ENTER key, which is required by the CAD system to end that command properly.

You may use any Lua technique to produce the command string which is subsequently passed on to the CAD system. Let x and y be coordinates in your current User Coordinate System. Then the .. operator in Lua chains text items together, converting numbers to letters as needed:

```
runCommand('._CIRCLE (list '..x..' '..y..') 0.25 ')
```

This latest example demonstrates how RailCOMPLETE script syntax, Lua and Lisp play together.

[Script commands reference manual](#)

For complete information, please consult the RailCOMPLETE Reference Manual which was bundled with your RailCOMPLETE installation and which is accessible from the **RC-Help** command.

(end of text)