openTCS
User's Guide

The openTCS developers

openTCS 4.7.0-SNAPSHOT

Table of Contents

Chapter 1. Introduction

1.1. Purpose of the software

openTCS is a control system software for (fleets of) automatic vehicles. It was primarily developed for the coordination of automated guided vehicles (AGV), but it is generally conceivable to use it with other automatic vehicles like mobile robots or quadrocopters, as openTCS controls the vehicles independent of their specific characteristics like track guidance system or load handling device. This is achieved by integrating vehicles into the system via pluggable drivers, similar to device drivers in operating systems.

1.2. System requirements

To run openTCS, a Java Runtime Environment (JRE) version 1.8 or later is required. (The directory `bin` of the installed JRE, for example `C:/Program Files/Java/jre1.8.0/bin`, should be included in the enviroment variable PATH to be able to use the included start scripts.)

1.3. Further documentation

If you intend to extend and customize openTCS, please also see the Developer's Guide and the JavaDoc documentation that is part of the openTCS distribution.

1.4. Questions and problem reports

If you have questions about this manual, the openTCS project or about using or extending openTCS, please contact the development team by using the discussion forums at http://sourceforge.net/projects/opentcs/ or by sending an e-mail to info@opentcs.org.

If you encounter technical problems using openTCS, please remember to include enough data in your problem report to help the developers help you, e.g.:

- The applications' log files, contained in the subdirectory `log/` of both the kernel and the plant overview application
- The plant model you are working with, contained in the subdirectory `data/` of the kernel and/or plant overview application

Chapter 2. System overview

2.1. System components and structure

openTCS consists of the following components running as separate processes and working together in a client-server architecture:

- Kernel (server process), running vehicle-independent strategies and drivers for controlled vehicles
- Clients
  - Plant overview for modelling and visualizing the plant model
  - Arbitrary clients for comunicating with other systems, e.g. for process control or warehouse management

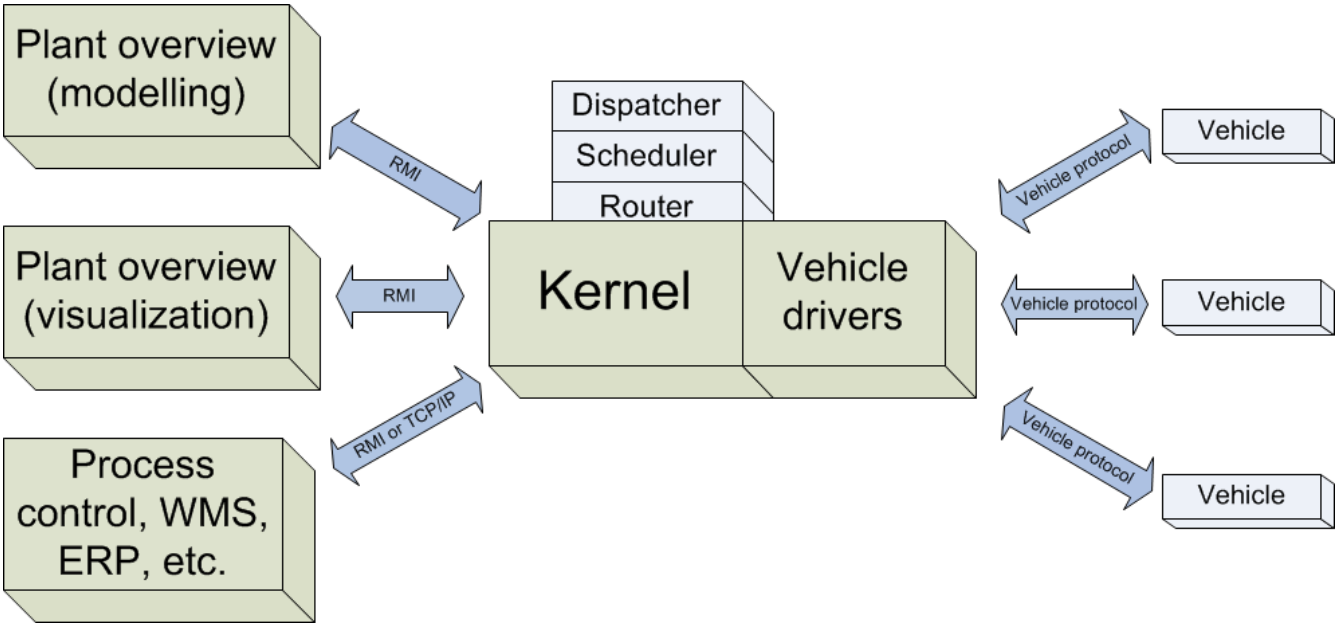

Figure 1. The architecture of openTCS

The purpose of the openTCS kernel is to provide an abstract driving model of a transportation system/plant, to manage transport orders and to compute routes for the vehicles. Clients can communicate with this server process to, for instance, modify the plant model, to visualize the driving course and the processing of transport orders and to create new transport orders. For user interaction, the kernel provides a graphical user interface titled Kernel Control Center.

Three major strategy modules within the kernel implement processing of transport orders:

- A dispatcher that decides which transport order should be processed by which vehicle. Additionally, it needs to decide what vehicles should do in certain situation, e.g. when there aren't any transport orders or when a vehicle is running low on energy.
- A router which finds optimal routes for vehicles to reach their destinations.
- A scheduler that manages resource allocations for traffic management, i.e. to avoid vehicles crashing into each other.

The openTCS distribution comes with default implementations for each of these strategies. These implementations can be easily replaced by a developer to adapt to environment-specific requirements.

The driver framework that is part of the openTCS kernel manages communication channels and associates vehicle drivers with vehicles. A vehicle driver is an adapter between kernel and vehicle and translates each vehicle-specific communication protocol to the kernel's internal communication schemes and vice versa. Furthermore, a driver may offer low-level functionality to the user via the kernel's graphical user interface, e.g. manually sending telegrams to the associated vehicle. By using suitable vehicle drivers, vehicles of different types can be managed simultaneously by a single openTCS instance.

The plant overview client that is part of the openTCS distribution allows editing of plant models, which can be loaded into the kernel. This includes, for instance, the definition of load-change stations, driving tracks and vehicles. In the kernel's plant operation mode, the plant overview client is used to display the transportation system's general state and any active transport processes, and to create new transport orders interactively.

Other clients, e.g. to control higher-level plant processes, can be implemented and attached. For Java clients, the openTCS kernel provides an interface based on Java RMI (Remote Method Invocation). A host interface for creating transport orders using XML telegrams sent via TCP/IP connections is also available.

2.2. Elements of plant models

In openTCS, a plant model consists of the following elements:

- *Points* are logical mappings of discrete positions reported by a vehicle. In plant operation mode, vehicles move from one point to another in the model. A point can have one of the following types:
  - Halt position: Indicates a position at which a vehicle may halt temporarily, e.g. for executing an operation. The vehicle is also expected to report in when it arrives at such a position. It may not park here for longer than necessary, though. Halt position is the default type for points when modelling with the plant overview client.
  - Reporting position: Indicates a position at which a vehicle is expected to report in. Halting or even parking at such a position is not allowed. Therefore a route that only consists of reporting points will be unroutable because the vehicle is not able to halt at any position.
  - Park position: Indicates a position at which a vehicle may halt for longer periods of time when it is not processing orders. The vehicle is also expected to report in when it arrives at such a position.
- *Paths* are connections between points that are navigable for vehicles.
- *Locations* are places at which vehicles may execute special operations (change their load, charge their battery etc.). To be reachable for any vehicle in the model, a location needs to be linked to at least one point.
- *Vehicles* map real vehicles for the purpose of visualizing their positions and other characteristics.

Furthermore, there is an abstract element that is only used indirectly:

- *Location types* group stations and define operations that can be executed by vehicles at these stations.

The attributes of these elements that are relevant for the driving course model, e.g. the coordinates of a point or the length of a path, can be edited using the plant overview client.

2.2.1. Layout coordinates vs model coordinates

A point has two sets of coordinates: layout coordinates and model coordinates. The layout coordinates are merely intended for the graphical presentation in the plant overview client, while the model coordinates are data that a vehicle driver could potentially use or send to the vehicle it communicates with (e.g. if the vehicle needs the exact coordinates of a destination point for navigation). Both coordinate sets are not tied to each other per se, i.e. they may differ. This is to allow coordinates that the system works with internally to be different from the presentation; for example, you may want to provide a distorted view on the driving course simply because some paths in your plant are very long and you mainly want to view all points/locations closely together. Dragging points and therefore changing their position in the graphical presentation only affects the corresponding layout coordinates.

To synchronize the layout coordinates with the model coordinates or the other way around you have two options:

- Select **Actions** | **Copy model values to layout** or **Actions** | **Copy layout values to model** to synchronize them globally.
- Select a single layout element, right click it and select **Context menu** | **Copy model values to layout** or **Context menu** | **Copy layout values to model** to synchronize them only for the selected element.

2.2.2. Generic element properties

Furthermore, it is possible to define arbitrary additional attributes as key-value pairs for all driving course elements, which for example can be read and evaluated by vehicle drivers or client software. Both the key and the value can be arbitrary character strings. For example, a key-value pair `"IP address":"192.168.23.42"` could be defined for a vehicle in the model, stating which IP address is to be used to communicate with the vehicle; a vehicle driver could now check during runtime whether a value for the key `"IP address"` was defined, and if yes, use it to automatically configure the communication channel to the vehicle. Another use for these generic attributes can be vehicle-specific actions to be executed on certain paths in the model. If a vehicle should, for instance, issue an acoustic warning and/or turn on the right-hand direction indicator when currently on a certain path, attributes with the keys `"acoustic warning"` and/or `"right-hand direction indicator"` could be defined for this path and evaluated by the respective vehicle driver.

Chapter 3. Operating the system

3.1. Starting the system

To create or to edit the plant model of a transport system, the openTCS Plant Overview application has to be started in modelling mode. To use it as a transportation control system based on an existing plant model, it has to be started in plant operation mode. Starting a component is done by executing the respective Unix shell script (`*.sh`) or Windows batch file (`*.bat`). By manipulating the script and adding `-Dopentcs.initialmode=` followed by `operating` or `modelling` you can automatically start in the specific mode.

3.1.1. Starting in modelling mode

1. Start the plant overview client (`startPlantOverview.bat/.sh`) and select 'Modelling mode'.
2. The plant overview will start with a new, empty model, but you can also load a model from a file (**File | Load Model**) or the current kernel model (**File | Load current kernel model**). The latter option requires a running kernel that the Plant Overview client can connect to and that has loaded an existing plant model.
3. Use the graphical user interface of the plant overview client to create an arbitrary driving course for your respective application/project. How you can add elements like points, paths and vehicles to your driving course is explained in detail in Constructing a new plant model.

3.1.2. Starting in plant operation mode

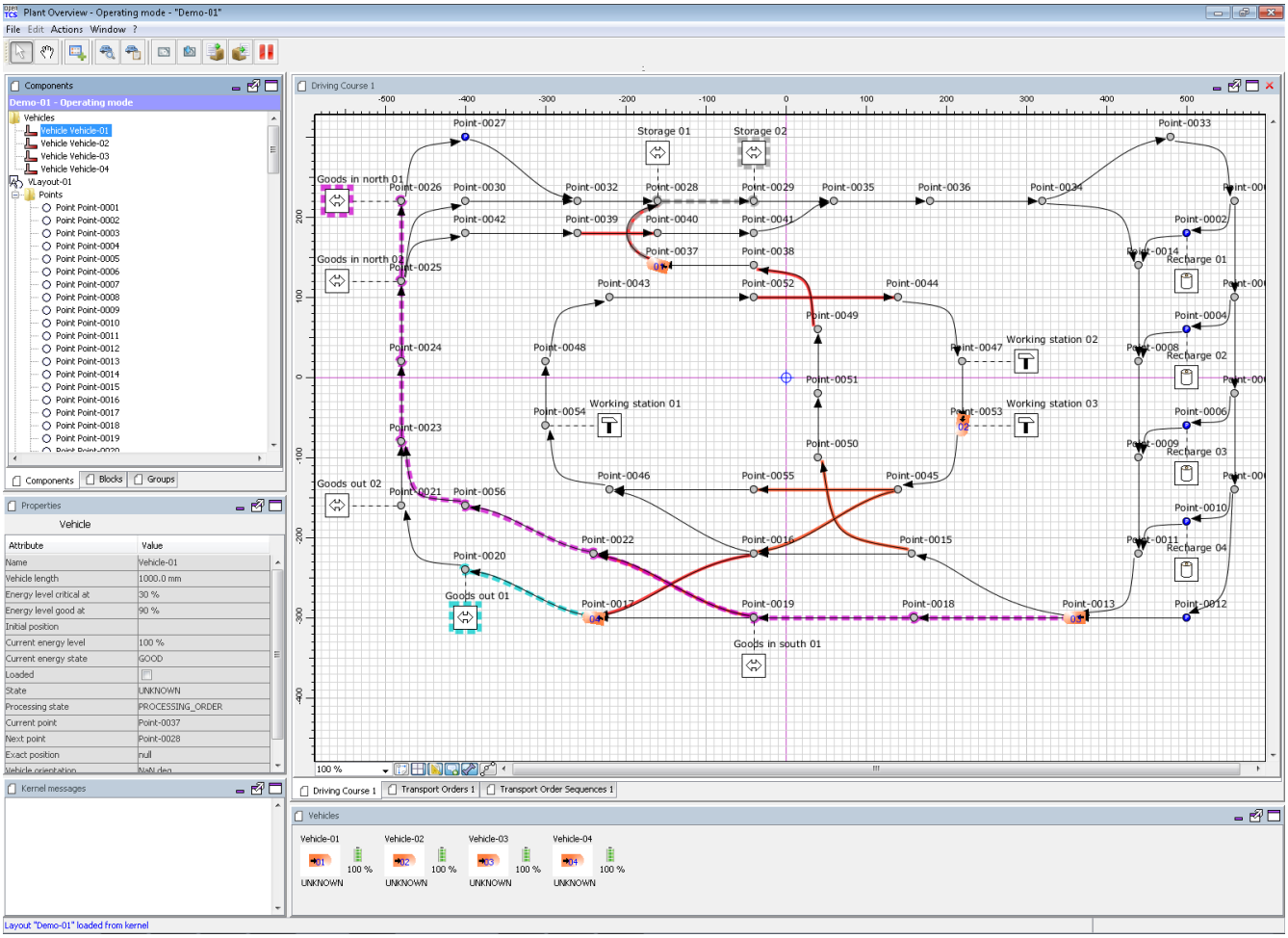

Figure 2. Plant overview client displaying plant model

1. Start the kernel (`startKernel.bat/.sh`).
    a. Select the saved model and plant operation mode in the dialog window shown and click OK.

    If this is your first time running the kernel, you need to persist the current plant model first. Start the kernel with an empty model instead and select **File | Persist model in the kernel** in the plant overview (also see <<Saving the plant model).

2. Start the plant overview client (`startPlantOverview.bat/.sh`) and select 'Operating mode'.
3. Select the tab **[Vehicle drivers]** in the kernel control center. Then select, configure and start driver for each vehicle in the model.
    a. The list on the left-hand side of the window shows all vehicles in the chosen model.
    b. A detailed view for a vehicle can be seen on the right-hand side of the driver panel after double-clicking on the vehicle in the list. The specific design of this detailed view depends on the driver associated with the vehicle. Usually, status information sent by the vehicle (e.g. current position and mode of operation) is displayed and low-level settings (e.g. for the vehicle's IP address) are provided here.
    c. Right-clicking on the list of vehicles shows a popup menu that allows to attach or detach drivers for selected vehicles.
    d. For a vehicle to be controlled by the system, a driver needs to be attached to the vehicle and enabled. (For testing purposes without real vehicles that could communicate with the

system, the so-called loopback driver can be used, which provides a virtual vehicle or simulates a real one.) How you attach and enable a vehicle driver is explained in detail in Configuring vehicle drivers.

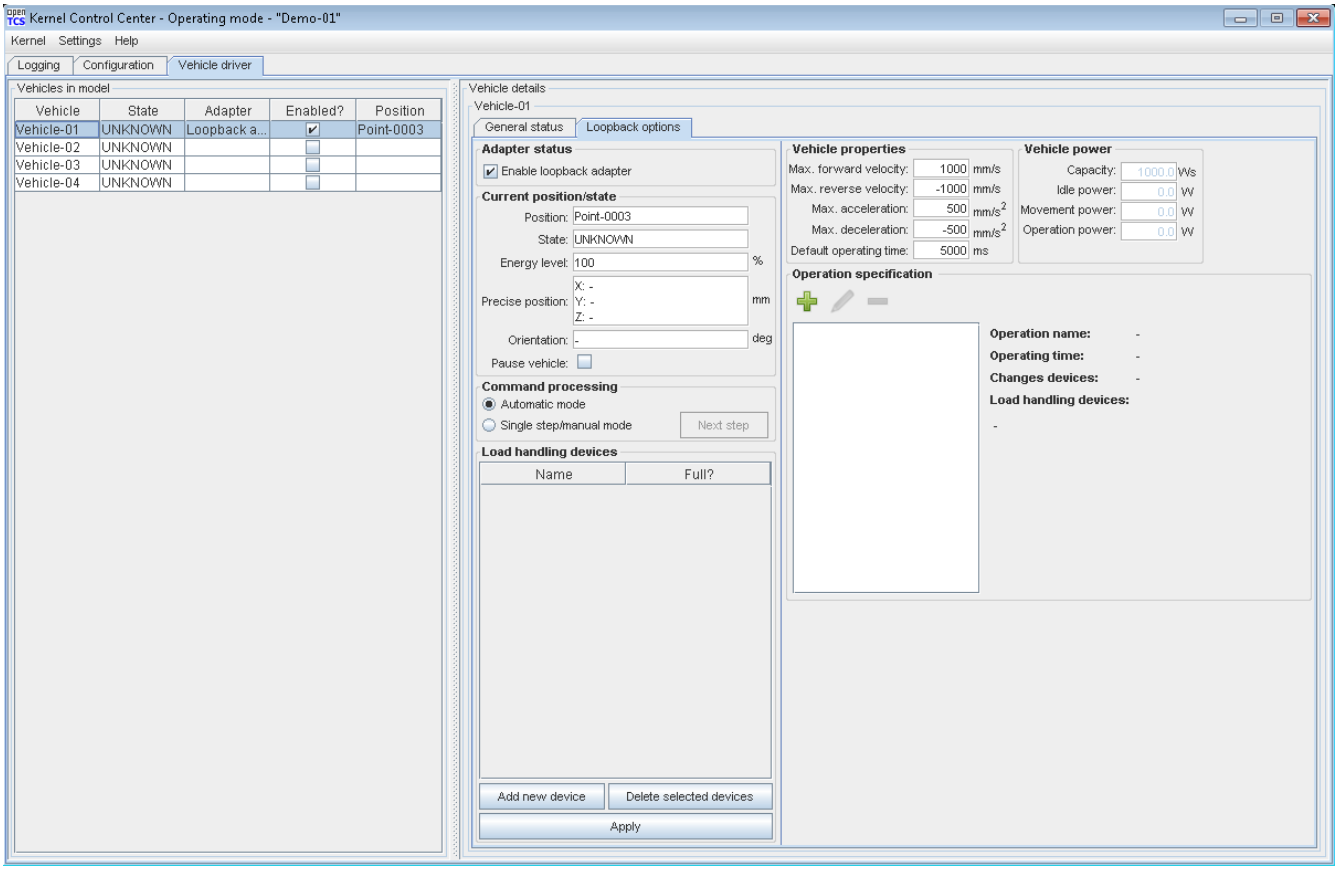

Figure 3. Driver panel with detailed view of a vehicle

3.2. Constructing a new plant model

These instructions roughly show how a new plant model is created and filled with driving course elements so that it can eventually be used in plant operation mode.

3.2.1. Starting components for plant modelling

1. Start the plant overview client (`startPlantOverview.bat/.sh`) and select 'Modelling mode'.
2. Wait until the graphical user interface of the plant overview client is shown.
3. You can now add driving course components to the empty model. Whenever you want to start over, select **File** | **New Model** from the main menu.

3.2.2. Adding elements to the plant model

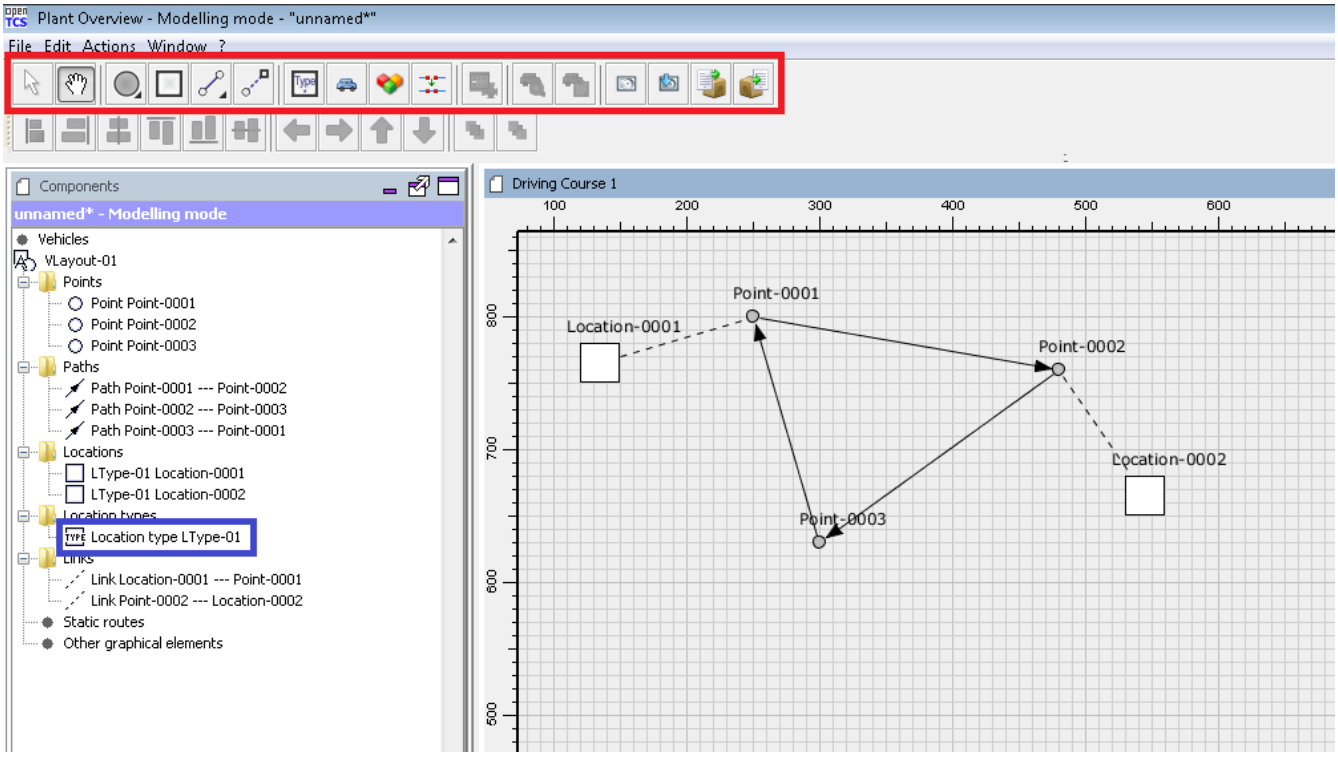

Figure 4. Control elements in the plant overview client (modelling mode)

1. Create three points by selecting the point tool from the driving course elements toolbar (see red frame in the screenshot above) and click on three positions on the drawing area.
2. Link the three points with paths to a closed loop by
   a. selecting the path tool by double-click.
   b. clicking on a point, dragging the path to the next point and releasing the mouse button there.
3. Create two locations by double-clicking the location tool and clicking on any two free positions on the drawing area. As a location type does not yet exist in the plant model, a new one is created implicitly when creating the first location, which can be seen in the tree view to the left of the drawing area.
4. Link the two locations with (different) points by
   a. double-clicking on the link tool.
   b. clicking on a location, dragging the link to a point and releasing the mouse button.
5. Create a new vehicle by clicking on the vehicle button in the course elements toolbar.
6. Define the allowed operations for vehicles at the newly created locations by
   a. selecting the locations' type in the tree view to the left of the drawing area (see blue frame in the screenshot above).
   b. clicking the value cell labelled `"Actions"` in the property window below the tree view.
   c. entering the allowed locations as arbitrary text in the dialog shown, for instance `"Load cargo"` and `"Unload cargo"`.
   d. Optionally, you can choose a symbol for locations of the selected type by editing the property `"Symbol"`.

You will not be able to create any transport orders and assign them to vehicles unless you create locations in your plant model, link these locations to points in the driving course and define the operations that vehicles may execute with the respective location types.

3.2.3. Saving the plant model

You have two options to save the model: on your local hard drive or in a running kernel instance the plant overview is connected to.

3.2.3.1. Saving the model locally

Select **File** | **Save Model** or **File** | **Save Model As...** and enter an arbitrary name for the model.

3.2.3.2. Persisting the model in a running kernel

Select **File** | **Persist model in the kernel** and your model will be persisted in the kernel, letting you switch to the operating mode. This, though, requires you to save it locally first. Note that the model that was previously persisted in the kernel will be replaced, as the kernel can only keep a single model at a time.

3.3. Operating the system

These instructions explain how the newly created model that was persisted in the kernel can be used in plant operation mode, how vehicle drivers are used and how transport orders can be created and processed by a vehicle.

3.3.1. Starting components for system operation

1. Start the kernel (`startKernel.bat/.sh`).
2. Wait until the dialog for selecting a plant model is shown.
3. Select the saved model you created, select plant operation mode, and click **[OK]**.
4. Start the plant overview client (`startPlantOverview.bat/.sh`), select 'Operating mode' and wait until its graphical user interface is shown.

3.3.2. Configuring vehicle drivers

1. Switch to the kernel control center window.
2. Associate the vehicle with the loopback driver by right-clicking on the vehicle in the vehicle list of the driver panel and selecting the menu entry **Driver** | **Loopback adapter (virtual vehicle)**.
3. Open the detailed view of the vehicle by double-clicking on the vehicle's name in the list.
4. In the detailed view of the vehicle that is now shown to the right of the vehicle list, select the tab **[Loopback options]**.
5. Enable the driver by ticking the checkbox **[Enable loopback adapter]** in the **[Loopback options]** tab or the checkbox in the **[Enabled?]** column of the vehicle list.

6. In the loopback options tab or in the vehicles list, select a point from the plant model to have the loopback adapter report this point to the kernel as the (virtual) vehicle's current position. (In a real-world application, a vehicle driver communicating with a real vehicle would automatically report the vehicle's current position to the kernel as soon as it is known.)
7. In the vehicle driver's loopback options tab, set the vehicle's state to `IDLE` to let the kernel know that the vehicle is now in a state that allows it to receive and process orders.
8. Switch to the plant overview client. An icon representing the vehicle should now be shown at the point on which you placed it using the loopback driver.
9. Right-click on the vehicle and select **Context menu** | **Dispatch Vehicle** to allow the kernel to dispatch the vehicle. The vehicle is then available for processing orders, which is indicated by a processing state `IDLE` in the property panel at the bottom left of the plant overview client's window. (You can revert this by right-clicking on the vehicle and selecting **Context menu** | **Withdraw TO and Disable Vehicle** in the context menu. The processing state shown is now `UNAVAILABLE` and the vehicle will not be dispatched for transport orders any more.)

3.3.3. Creating a transport order

To create a transport order, the plant overview client provides a dialog window presented when selecting **Actions** | **Transport Order** in the menu. Transport orders are defined as a sequence of destination locations at which actions are to be performed by the vehicle processing the order. You can select a destination location and action from a dropdown menu. You may also optionally select the vehicle intended to process this order. If none is explicitly selected, the control system automatically assigns the order to a vehicle according to its internal strategies - with the default strategy, it will pick the vehicle that will most likely finish the transport order the soonest. Furthermore, a transport order can be given a deadline specifying the point of time at which the order should be finished at the latest. This deadline will primarily be considered when there are multiple transport orders in the pool and openTCS needs to decide which to assign next.

To create a new transport order, do the following:

1. Select the menu entry **Actions** | **Transport Order**.
2. In the dialog shown, click the **[Add]** button and select a location as the destination and an operation which the vehicle should perform there. You can add an arbitrary number of destinations to the order this way. They will be processed in the given order.
3. After creating the transport order with the given destinations by clicking **[OK]**, the kernel will check for a vehicle that can process the order. If a vehicle is found, it is assigned the order immediately and the route computed for it will be highlighted in the plant overview client. The loopback driver simulates the vehicle's movement to the destinations and the execution of the operations.

3.3.4. Withdrawing transport orders using the plant overview client

A transport order can be withdrawn from a vehicle that is currently processing it. This can be done by right-clicking on the respective vehicle in the plant overview client and selecting **Context menu** | **Withdraw Transport Order**. The processing of the order will be cancelled and the vehicle (driver) will not receive any further drive orders. Processing of this transport order *cannot* be

resumed later. Instead, a new transport order will have to be created.

3.3.5. Continuous creation of transport orders

The plant overview client can easily be extended via custom plugins. As a reference, a simple load generator plugin is included which also serves as a demonstration of how the system looks like during operation here. Details about how custom plugins can be created and integrated into the plant overview client can be found in the developer's guide.

1. In the plant overview client, select **View** | **Plugins** | **Continuous load** from the menu.
2. Choose a trigger for creating new transport orders: New orders will either be created once only, or if the number of active orders in the system drops below a specified limit, or after a specified timeout has expired.
3. By using an order profile you may decide if the transport orders' destinations should be chosen randomly or if you want to choose them yourself.

   Using **[Create orders randomly]**, you define the number of transport orders that are to be generated at a time, and the number of destinations a single transport order should contain. Since the destinations will be selected randomly, the orders created might not necessarily make sense for a real-world system.

   Using **[Create orders according to definition]**, you can define an arbitrary number of transport orders, each with an arbitrary number of destinations and properties, and save and load your list of transport orders.

4. Start the order generator by activating the corresponding checkbox at the bottom of the **[Continuous load]** panel. The load generator will then generate transport orders according to its configuration until the checkbox is deactivated or the panel is closed.

3.3.6. Statistics reports about transport orders and vehicles

While running in plant operation mode, the openTCS kernel collects data about processed, finished and failed transport orders as well as busy and idle vehicles. It writes this data to log files in the `log/statistics/` subdirectory. To see a basic statistics report for the order processing in a plant operation session, you can use another plugin for the plant overview client that comes with the openTCS distribution:

1. In the plant overview client, select **View** | **Plugins** | **Statistics** from the menu.
2. Click the **[Read input file]** button and select a log file from `log/statistics/` in the kernel application's directory.
3. The panel will then show an accumulation of the data collected in the statistics log file you opened.

As the steps above should indicate, the statistics plugin currently does not provide a live view on statistical data in a running plant operation session. The report is an offline report that can be generated only after a plant operation session has ended. Future versions of openTCS may include a live report plugin that collects data directly from the openTCS kernel instead of reading the data from a log file.

3.3.7. Removing a vehicle from a running system

There may be situations in which you want to remove a single vehicle from a system, e.g. because the vehicle temporarily cannot be controlled by openTCS due to a hardware defect that has to be dealt with first. The following steps will ensure that no further transport orders are assigned to the vehicle and that the resources it might still be occupying are freed for use by other vehicles.

1. In the plant overview client, right-click on the vehicle and select **Context menu** | **Withdraw TO and Disable Vehicle** to disable the vehicle for transport order processing.
2. In the kernel control center, disable the vehicle's driver by unticking the checkbox **[Enable loopback adapter]** in the **[Loopback options]** tab or the checkbox in the **[Enabled?]** column of the vehicle list.
3. In the kernel control center, right-click on the vehicle in the vehicle list and select **Context menu** | **Reset vehicle position** to free the point in the driving course that the vehicle is occupying.

Chapter 4. Default strategies

openTCS comes with a default implementation for each of the strategy modules. These implementations can easily be replaced to adapt to project-specific requirements. (See developer's guide.)

4.1. Default dispatcher

When a transport order becomes dispatchable, the dispatcher needs to decide which vehicle should process it. To make this decision, the default dispatcher takes the following steps:

1. The default dispatcher checks which vehicles are available and not currently processing a transport order. If there are no such vehicles, the incoming transport order is postponed without a vehicle assignment.
2. The default dispatcher requests the potential routes from the router for all vehicle candidates in case they were selected for the incoming transport order. If no viable route is available for any of the vehicles, the incoming transport order is postponed without a vehicle assignment.
3. From all vehicles with viable routes, the one with the lowest routing costs is assigned to the incoming transport order.

When a vehicle becomes available, the dispatcher needs to decide which transport order it should process. To make this decision, the default dispatcher takes the following steps:

1. If the vehicle's energy level is not *critical* and transport orders are available, the default dispatcher will select the most urgent one (according to its deadline) for which a route can be computed.
2. If the vehicle's energy level is *degraded* and automatic recharging is enabled, an order is created for the vehicle to recharge at a recharging location.
3. If the vehicle is not currently at a parking position and there are unoccupied parking positions, an order is created for the vehicle to move to one of them.
4. If none of the previous steps assigned an order to the vehicle, it is left at its current position.

When sending a vehicle to a recharge location or parking position, the closest (according to the router) unoccupied position is selected by default. It is possible to assign fixed positions to vehicles instead, by setting properties with the following keys on them:

- `tcs:preferredParkingPosition`: Expected to be the name of a point in the model. If this point is already occupied, the closest unoccupied parking position (if any) is selected instead.
- `tcs:assignedParkingPosition`: Expected to be the name of a point in the model. If this point is already occupied, the vehicle is not sent to any other parking position, i.e. remains where it is. Takes precedence over `tcs:preferredParkingPosition`.
- `tcs:preferredRechargeLocation`: Expected to be the name of a location. If this location is already occupied, the closest unoccupied recharging location (if any) is selected instead.
- `tcs:assignedRechargeLocation`: Expected to be the name of a location. If this location is already occupied, the vehicle is not sent to any other recharging location. Takes precedence over

`tcs:preferredRechargeLocation`.

4.2. Default router

The default router finds the cheapest route from one position in the driving course to another one. (It uses an implementation of Dijkstra's algorithm to do that.) It takes into account paths that have been locked, but not positions and/or assumed future behaviour of other vehicles. As a result, it does not route around slower or stopped vehicles blocking the way.

The cost function used for evaluating the edges of the graph can be configured — see Default router configuration entries. The default cost function for a path simply evaluates to the path's length, so the cheapest route by default is the shortest one.

It is possible to treat vehicles in a plant differently when computing their routes. This may be desirable if they have different characteristics and actually have different optimal routes through the driving course. For this to work, the paths in the model or the cost function used need to reflect reflect this difference. This isn't done by default — the default router computes routes for all vehicles the same way unless told otherwise. To let the router know that it should compute routes for a vehicle separately, set a property with the key `tcs:routingGroup` to an integer value other than zero. (Vehicles that have the same integer value set share the same routing table, and zero is the default value for all vehicles.)

4.3. Default scheduler

The default scheduler implements a simple strategy for traffic management. It does this by allowing only mutually exclusive use of resources in the plant model (points and paths, primarily):

- When an allocation of a set of resources for a vehicle is requested, the scheduler expands this set to the effective resource set (in case any resources are part of a block area) and checks whether the allocation can be granted immediately. If yes, the allocation is made. If not, the allocation is queued for later.
- Whenever resources are freed (e.g. when a vehicle has finished its movement to the next point and the vehicle driver reports this to the kernel), the allocations waiting in the queue are checked (in the order the requests happened). Any allocations that can now be made are made. Allocations that cannot be made are kept waiting.

This strategy ensures that resources are used when they are available. It does not, however, strictly ensure fairness/avoid starvation: Vehicles waiting for allocation of a large resource set may theoretically wait forever if other vehicles can keep allocating subsets of those resources continuously. Such situations are likely a hint at problems in the plant model graph's topology, which is why this deficiency is considered acceptable for the default implementation.

Chapter 5. Configuring openTCS

5.1. Kernel configuration

The kernel application reads its configuration data from two files — `config/opentcs-kernel-defaults.properties` and `config/opentcs-kernel.properties`. (Both files must exist.) It is good practice to leave the first one untouched and set overriding values and project-specific configuration data in the second one.

5.1.1. Kernel application configuration entries

The kernel application itself can be configured using the following configuration entries:

Table 1. Configuration options with prefix 'kernelapp'

Key	Type	Description
selectModelOnStartup	Boolean	Whether to show the model selection dialog on startup.
loadModelOnStartup	Boolean	Whether to automatically load the kernel's model on startup.
autoAttachDriversOnStartup	Boolean	Whether to automatically attach drivers on startup.
autoEnableDriversOnStartup	Boolean	Whether to automatically enable drivers on startup.
saveModelOnTerminateModelling	Boolean	Whether to implicitly save the model when leaving modelling state.
saveModelOnTerminateOperating	Boolean	Whether to implicitly save the model when leaving operating state.
updateRoutingTopologyOnPathLockChange	Boolean	Whether to implicitly update the router's topology when a path is (un)locked.

5.1.2. Control center configuration entries

The kernel's control center GUI can be configured using the following configuration entries:

Table 2. Configuration options with prefix 'controlcenter'

Key	Type	Description
enable	Boolean	Whether the kernel control center GUI should be enabled on startup. (EXPERIMENTAL)
language	String	The application's current language. Valid values: 'English', 'German'

Key	Type	Description
loggingAreaCapacity	Integer	The maximum number of characters in the logging text area.

5.1.3. Order pool configuration entries

The kernel's transport order pool can be configured using the following configuration entries:

Table 3. Configuration options with prefix 'orderpool'

Key	Type	Description
sweepAge	Integer	The minimum age of orders to remove in a sweep (in ms).
sweepInterval	Long	The interval between sweeps (in ms).

5.1.4. Vehicle communication configuration entries

General communication with vehicles can be configured using the following configuration entries:

Table 4. Configuration options with prefix 'vehicles'

Key	Type	Description
ignoreUnknownReportedPositions	Boolean	Whether to ignore unknown positions reported by a vehicle. If not ignored, unknown positions reset the vehicle's position in the course model.

5.1.5. Default dispatcher configuration entries

The default dispatcher can be configured using the following configuration entries:

Table 5. Configuration options with prefix 'defaultdispatcher'

Key	Type	Description
parkIdleVehicles	Boolean	Whether to automatically create parking orders idle vehicles.
rechargeIdleVehicles	Boolean	Whether to automatically create recharge orders for idle vehicles.
assignRedundantOrders	Boolean	Whether orders to the current position with no operation should be assigned.
dismissUnroutableTransportOrders	Boolean	Whether unroutable incoming transport orders should be marked as UNROUTABLE.
idleVehicleRedispatchingInterval	Integer	The interval between redispatching of vehicles.

5.1.6. Default router configuration entries

The default router can be configured using the following configuration entries:

Table 6. Configuration options with prefix 'defaultrouter'

Key	Type	Description
routeToCurrentPosition	Boolean	Whether to compute a route even if the vehicle is already at the destination.

The shortest path algorithm can be configured using the following configuration entries:

Table 7. Configuration options with prefix 'defaultrouter.shortestpath'

Key	Type	Description
algorithm	Strings	The routing algorithm to be used. Valid values: 'DIJKSTRA': Routes are computed using Dijkstra's algorithm. 'FLOYD_WARSHALL': Routes are computed using Floyd-Warshall algorithm.
edgeEvaluators	List of strings	The types of route evaluators/cost factors to be used. Results of multiple evaluators are added up. Valid values: 'DISTANCE': A route's cost is the sum of the lengths of its paths. 'TRAVELTIME': A route's cost is the vehicle's expected driving time to the destination. 'TURNS': A route's cost is the number of turns/direction changes on it. 'EXPLICIT': A route's cost is the sum of the explicitly given costs of its paths.

5.1.7. RMI kernel interface configuration entries

The kernel's RMI interface can be configured using the following configuration entries:

Table 8. Configuration options with prefix 'rmikernelinterface'

Key	Type	Description
registryHost	String	The host name/IP address of the RMI registry. If 'localhost' and not running already, a RMI registry will be started.
registryPort	Integer	The TCP port of the RMI.
remoteKernelPort	Integer	The TCP port of the remote kernel.
clientSweepInterval	Long	The interval for cleaning out inactive clients (in ms).

5.1.8. XML host interface configuration entries

The kernel's XML-based host interface can be configured using the following configuration entries:

Table 9. Configuration options with prefix 'xmlhostinterface'

Key	Type	Description
ordersServerPort	Integer	The TCP port on which to listen for incoming order connections.
ordersIdleTimeout	Integer	The time (in ms) after which idle connections are closed.
ordersInputLimit	Integer	The maximum number of bytes read from sockets before closing the connection.
statusServerPort	Integer	The TCP port on which to listen for incoming status channel connections.
statusMessageSeparator	String	A string to be used for separating subsequent status messages in the stream.

5.1.9. Virtual vehicle configuration entries

The virtual vehicle (loopback communication adapter) can be configured using the following configuration entries:

Table 10. Configuration options with prefix 'virtualvehicle'

Key	Type	Description
commandQueueCapacity	Integer	The adapter's command queue capacity.
rechargeOperation	String	The string to be treated as a recharge operation.
simulationTimeFactor	Double	The simulation time factor. 1.0 is real time, greater values speed up simulation.
profilesMaxFileSize	Long	The maximum allowed size (in bytes) for a profiles description file.

5.2. Plant Overview configuration

The plant overview application reads its configuration data from two files — `config/opentcs-plantoverview-defaults.properties` and `config/opentcs-plantoverview.properties`. (Both files must exist.) It is good practice to leave the first one untouched and set overriding values and project-specific configuration data in the second one.

5.2.1. Plant Overview application configuration entries

The plant overview application itself can be configured using the following configuration entries:

Table 11. Configuration options with prefix 'plantoverviewapp'

Key	Type	Description
language	String	The plant overview application’s locale. Valid values: 'English', 'German'
initialMode	String	The plant overview application’s mode on startup. Valid values: 'MODELLING', 'OPERATING', 'ASK'
frameMaximized	Integer	Whether the GUI window should be maximized on startup.
frameBoundsWidth	Integer	The GUI window’s configured width in pixels.
frameBoundsHeight	Integer	The GUI window’s configured height in pixels.
frameBoundsX	Integer	The GUI window’s configured x-coordinate on screen in pixels.
frameBoundsY	Integer	The GUI window’s configured y-coordinate on screen in pixels.
connectionBookmarks	List of <hostname:port>	The configured connection bookmarks.
locationThemeClass	Class name	The name of the class to be used for the location theme. Must be a class extending org.opentcs.components.plantoverview.LocationTheme
vehicleThemeClass	Class name	The name of the class to be used for the vehicle theme. Must be a class extending org.opentcs.components.plantoverview.VehicleTheme
ignoreVehiclePrecisePosition	Boolean	Whether reported precise positions should be ignored displaying vehicles.
ignoreVehicleOrientationAngle	Boolean	Whether reported orientation angles should be ignored displaying vehicles.

Chapter 6. Advanced usage examples

6.1. Configuring automatic startup

1. To have the kernel automatically load a plant model on startup, set the kernel application's configuration parameter `kernelapp.loadModelOnStartup` to `true`.
2. To suppress the startup dialog that is normally shown by the kernel application on startup, set its configuration parameter `kernelapp.selectModelOnStartup` to `false`.
3. For automatic attachment of vehicle drivers to vehicles on startup, set the kernel application's configuration parameter `kernelapp.autoAttachDriversOnStartup` to `true`.
4. To automatically enable vehicle drivers on startup, set the kernel application's configuration parameter `kernelapp.autoEnableDriversOnStartup` to `true`.

6.2. Automatically selecting a specific vehicle driver on startup

Automatic attachment of vehicle drivers by default works as follows: The kernel asks every available vehicle driver if it can attach to a given vehicle and selects the first one that can. It asks the loopback driver last, as that one is always available and can attach to any vehicle, but should not prevent actual vehicle drivers to be attached. As a result, if there is only one driver for your vehicle(s), you usually do not have to do anything for it to be selected.

In some less common cases, you may have multiple vehicle drivers registered with the kernel that can all attach to the vehicles in your plant model. To automatically select a specific driver in such cases, set a property with the key `tcs:preferredAdapterClass` on the vehicles, with its value being the name of the Java class implementing the driver's adapter factory. (If you do not know this class name, ask the developer who provided the vehicle driver to you for it.)

6.3. Initializing a virtual vehicle's position automatically on startup

Even during the development or simulation phase of a project, repeatedly setting a virtual vehicle's initial position manually via the loopback driver's user interface can be tiresome and possibly error-prone. To automatically set a vehicle's initial position when the loopback driver is enabled, you can store the initial position in the respective plant model by doing the following:

1. Start the plant overview client in modelling mode.
2. In the plant overview client's tree view of the plant model, select a vehicle.
3. In the table showing the vehicle's properties, click into the value field labelled **[Initial position]**. In the dialog shown, select the initial position for this vehicle from the model's list of points.
4. Save the model and persist it in the kernel as described in Saving the plant model.

Whenever you attach the loopback driver to the modified vehicle after doing this, it will automatically report the vehicle's position to be the one that you selected.

6.4. Running kernel and plant overview on separate systems

The kernel and the plant overview client communicate via Java's Remote Method Invocation (RMI) mechanism. This makes it possible to run the kernel and the plant overview client on separate systems, as long as a network connection between these systems exists and is usable.

By default, the plant overview client is configured to connect to a kernel running on the same system. To connect it to a kernel running on a remote system, e.g. on a host named myhost.example.com, set the plant overview application's configuration parameter `plantoverviewapp.connectionBookmarks` to `myhost.example.com:1099`. The configuration value can be a comma-separated list of host:port pairs. The plant overview will automatically try to connect to the first host in the list. If that fails, it will show a dialog to select an entry or enter a different address.

6.5. Configuring automatic parking and recharging

By default, idle vehicles remain where they are after processing their last transport order. You can change this in the kernel's configuration file:

- To order vehicles to charging locations automatically, set the configuration parameter `defaultdispatcher.rechargeIdleVehicles` to `true`. The default dispatcher will then look for locations at which the idle vehicle's recharge operation is possible and create orders to send it to such a location (if unoccupied). (Note that the string used for the operation is driver-specific.)
- To order vehicles to parking positions automatically, set the configuration parameter `defaultdispatcher.parkIdleVehicles` to `true`. The default dispatcher will then look for unoccupied parking positions and create orders to send the idle vehicle there.

6.6. Selecting the cost factors used for routing

The default router can evaluate the costs for routes based on different factors. You can select which factors should be taken into account by setting the configuration parameter `defaultrouter.routeEvaluators` in the kernel's configuration file to one or more of the following key words (separated by commas):

- `HOPS`: Routing costs are measured by the number of hops/paths travelled along the route.
- `DISTANCE`: Routing costs are measured by the sum of the lengths of paths travelled.
- `TRAVELTIME`: Routing costs are measured by the sum of the times required for travelling each path on a route. The travel times are computed using the length of the respective path and the maximum speed with which a vehicle may move on it.
- `TURNS`: Routing costs are measured by the number of changes of the vehicle's orientation on a route. To specify an orientation for a vehicle on a path, set a property on that path with a key of `"tcs:travelOrientation"` and an arbitrary string as the value. If, and only if, the values of

subsequent paths on a route differ, a penalty is added to the costs of that route. This penalty can be configured by setting the configuration entry `defaultrouter.turnCosts` to an integer value (default: 5000).

- `EXPLICIT`: Routing costs are measured by the sum of the costs explicitly specified by the modelling user. Explicit costs can be specified for every single path in the model using the plant overview client. (Select a path and set its **[Costs]** property to an arbitrary integer value.)

When specifying more than one of these key words, the respective costs computed are added up. For example, when set to `"DISTANCE, TURNS"`, costs for routes are computed as the sum of the paths' lengths and the penalties for orientation changes. If none of these entries is set, costs for routes are computed by the paths' lengths by default (`DISTANCE`).

6.7. Configuring order pool cleanup

By default, openTCS checks every minute for finished or failed transport orders that are older than 24 hours. These orders are removed from the pool. To customize this behaviour, do the following:

1. Set the configuration entry `orderpool.sweepInterval` to a value according to your needs. The default value is 60.000 (milliseconds, corresponding to an interval of one minute).
2. Set the configuration entry `orderpool.sweepAge` to a maximum age of finished orders according to your needs. The default value is 86.400.000 (milliseconds, corresponding to 24 hours that a finished order should be kept in the pool).

6.8. Using model element properties for project-specific data

Every object in the plant model - i.e. points, paths, locations, location types and vehicles - can be augmented with arbitrary project-specific data that can be used, e.g. by vehicle drivers, custom client applications, etc.. Possible uses for such data could be informing the vehicle driver about additional actions to be performed by a vehicle when moving along a path in the model (e.g. flashing direction indicators, displaying a text string on a display, giving an acoustic warning) or controlling the behaviour of peripheral systems (e.g. automatic fire protection gates).

The data can be stored in properties, i.e. key-value pairs attached to the model elements, where both the key and the corresponding value are text strings. These key-value pairs can be created and edited using the plant overview client: Simply select the model element you want to add a key-value pair to and click into the value field labelled **[Miscellaneous]** in the properties table. In the dialog shown, set the key-value pairs you need to store your project-specific information.

For your project-specific key-value pairs, you may specify arbitrary keys. openTCS itself will not make any use of this data; it will merely store it and provide it for custom vehicle drivers and/or other extensions. You should, however, not use any keys starting with `"tcs:"` for storing project-specific data. Any keys with this prefix are reserved for official openTCS features, and using them could lead to collisions.