openTCS

User's Guide

The openTCS developers

openTCS 4.20.0

Table of Contents

Chapter 1. Introduction

1.1. Purpose of the software

openTCS is a control system/fleet management software for automatic vehicles. It was primarily developed for the coordination of automated guided vehicles (AGV) performing transportation tasks e.g. in a production plant. However, it can be used with other automatic vehicles like mobile robots or quadrocopters, too. openTCS controls the vehicles independent of their specific characteristics like navigation principle/track guidance system or load handling device. It can manage vehicles of different types (and performing different tasks) at the same time. This is achieved by integrating vehicles into the system via pluggable drivers, similar to device drivers in operating systems.

1.2. System requirements

openTCS does not come with any specific hardware requirements. CPU power and RAM capacity highly depend on the use case, e.g. the size and complexity of the driving course and the number of vehicles managed. Some kind of networking hardware — in most cases simply a standard Ethernet controller — is required for communicating with the vehicles (and possibly other systems, like a warehouse management system).

To run openTCS, a Java Runtime Environment (JRE) version 1.8 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 mailing list at http://sourceforge.net/projects/opentcs/.

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
  - Kernel control center for controlling and monitoring the kernel, e.g. providing a detailed view of vehicles/their associated drivers
  - Arbitrary clients for comunicating with other systems, e.g. for process control or warehouse management

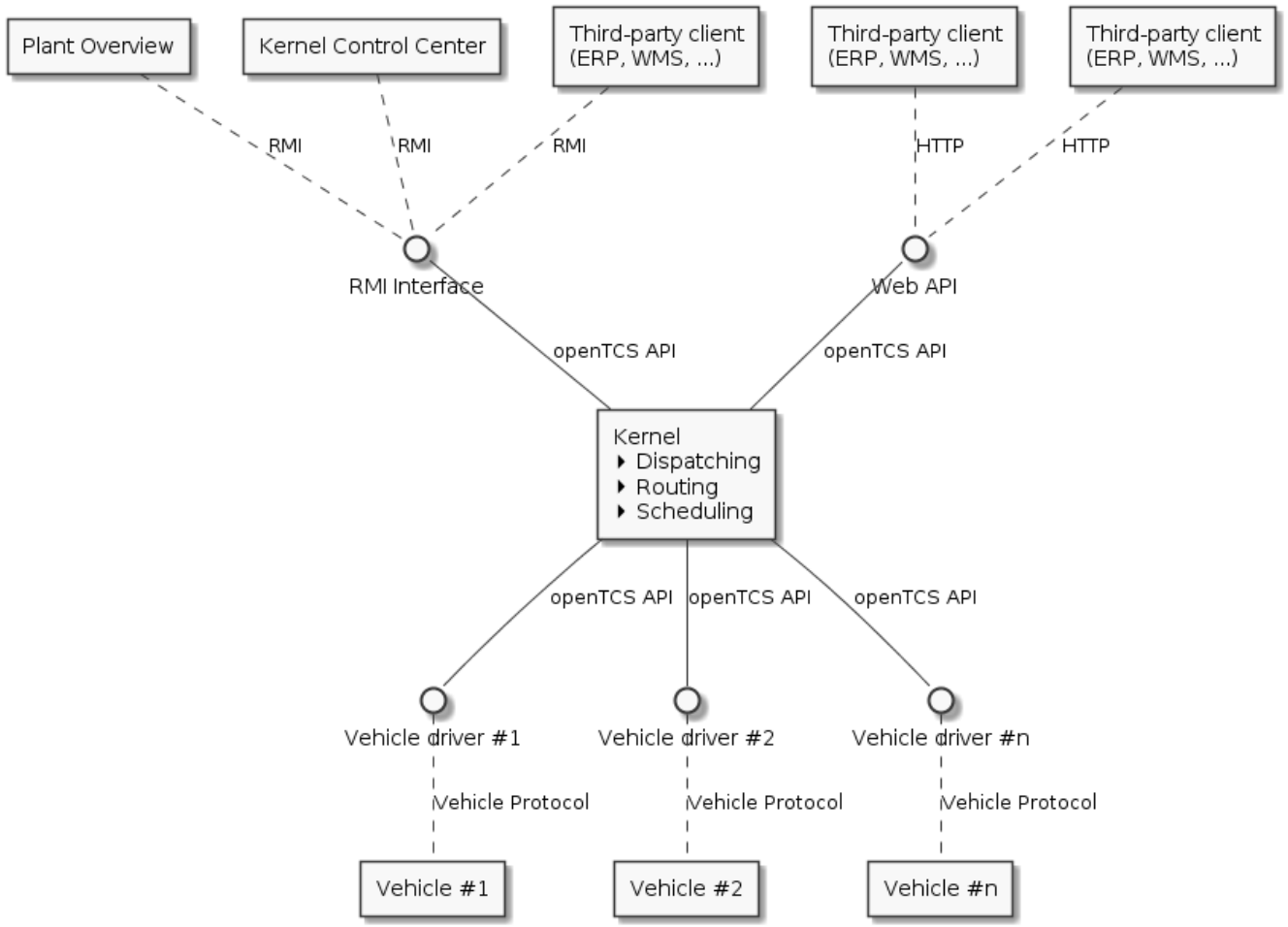

Figure 1. openTCS system overview

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. Additionally there's also a standalone/remote version of the Kernel Control Center.

The Kernel Control Center GUI provided by the kernel application is deprecated and scheduled for removal with version 5.0. A separate Kernel Control Center application is available and should be used instead.

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.

The kernel control center client that is part of the openTCS distribution allows controlling and monitoring the kernel. Part of that is assigning vehicle drivers to vehicles and controlling them by enabling the communication and monitoring them by displaying vehicle state information, for instance.

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. Additionally, openTCS provides a web API for creating and withdrawing transport orders and retrieving transport order status updates.

2.2. Plant model elements

In openTCS, a plant model consists of a set of the following elements. The attributes of these

elements that are relevant for the plant model, e.g. the coordinates of a point or the length of a path, can be edited using the plant overview client (in modelling mode).

2.2.1. Point

Points are logical mappings of discrete vehicle positions in the driving course. In plant operation mode, vehicles are ordered (and thus move) from one point to another in the model. A point carries the following attributes:

- A *type*, which is one of these three:
  - *Halt position*: Indicates a position at which a vehicle may halt temporarily while processing an order, e.g. for executing an operation. The vehicle is expected to report in when it arrives at such a position. It may not remain 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 *only*. Vehicles will not be ordered to a reporting position, and 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.
- A *position*, i.e. the point's coordinates in the plant's coordinate system.
- A *vehicle orientation angle*, which expresses the vehicle's assumed/expected orientation while it occupies the point.

2.2.1.1. 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. Path

Paths are connections between points that are navigable for vehicles. A path's main attributes, next to its source and destination point, are:

- Its *length*, which may be a relevant information for a vehicle in plant operation mode. Depending on the router configuration, it may also be used for computing routing costs/finding an optimal route to a destination point.
- A *maximum velocity* and *maximum reverse velocity*, which may be a relevant information for a vehicle in plant operation mode. Depending on the router configuration, it may also be used for computing routing costs/finding an optimal route to a destination point.
- A *routing cost*, which is an explicit, unitless value. Depending on the router configuration, it may be used for computing routing costs/finding an optimal route to a destination point.
- A *locked* flag, which, when set, tells the router that the path may not be used when computing routes for vehicles.

2.2.3. Location

Locations are markers for points at which vehicles may execute special operations (load or unload cargo, charge their battery etc.). A location's attributes are:

- Its *type*, basically defining which operations are allowed at the location — see Location type.
- A set of *links* to points that the location can be reached from. To be of any use for vehicles in the plant model, a location needs to be linked to at least one point.

2.2.4. Location type

Location types are abstract elements that group locations. A location type has only one relevant attribute:

- A set of *allowed operations*, defining which operations a vehicle may execute at locations of this type.

2.2.5. Vehicle

Vehicles map physical vehicles for the purpose of communicating with them and visualizing their positions and other characteristics. A vehicle provides the following attributes:

- A *critical energy level*, which is the threshold below which the vehicle's energy level is considered critical. This value may be used at plant operation time to decide when it is crucial to recharge a vehicle's energy storage.
- A *good energy level*, which is the threshold above which the vehicle's energy level is considered good. This value may be used at plant operation time to decide when it is unnecessary to recharge a vehicle's energy storage.
- A *fully recharged energy level*, which is the threshold above which the vehicle is considered being fully recharged. This value may be used at plant operation time to decide when a vehicle should stop charging.

- A *sufficiently recharged energy level*, which is the threshold above which the vehicle is considered sufficiently recharged. This value may be used at plant operation time to decide when a vehicle may stop charging.
- A *maximum velocity* and *maximum reverse velocity*. Depending on the router configuration, it may be used for computing routing costs/finding an optimal route to a destination point.
- An *integration level*, indicating how far the vehicle is currently allowed to be integrated into the system. A vehicle can be
  - ...*ignored*: The vehicle and its reported position will be ignored completely, thus the vehicle will not be displayed in the plant overview. The vehicle is not available for transport orders.
  - ...*noticed*: The vehicle will be displayed at its reported position in the plant overview, but no resources will be allocated in the system for that position. The vehicle is not available for transport orders.
  - ...*respected*: The resources for the vehicle's reported position will be allocated. The vehicle is not available for transport orders.
  - ...*utilized*: The vehicle is available for transport orders and will be utilized by the openTCS.
- A set of *allowed transport order types*, which are strings used for filtering transport orders (by their type) that are allowed to be assigned to the vehicle. Also see Transport order.
- A *route color*, which is the color used for visualizing the route the vehicle is taking to its destination.

2.2.6. Block

Blocks (or block areas) are areas for which special traffic rules may apply. They can be useful to prevent deadlock situations, e.g. at path intersections or dead ends. A block has two relevant attributes:

- A set of *members*, i.e. resources (points, paths and/or locations) that the block is composed of.
- A *type*, which determines the rules for entering a block:
  - *Single vehicle only*: The resources aggregated in this block can only be used by a single vehicle at the same time. This is the default type for blocks when modelling with the plant overview client.
  - *Same direction only*: The resources aggregated in this block can be used by multiple vehicles at the same time, but only if they traverse the block in the same direction.

The direction in which a vehicle traverses a block is determined using the first allocation request containing resources that are part of the block — see Default scheduler. For the requested resources (usually a point and a path) the path is checked for a property with the key `tcs:blockEntryDirection`. The property's value may be an arbitrary character string (including the empty string). If there is no such property the path's name is being used as the direction.

2.3. Plant operation elements

Transport orders and order sequences are elements that are available only at plant operation time. Their attributes are primarily set when the respective elements are created.

2.3.1. Transport order

A transport order is a parameterized sequence of movements and operations to be processed by a vehicle. When creating a transport order, the following attributes can be set:

- A sequence of *destinations* that the processing vehicle must process (in their given order). Each destination consists of a location that the vehicle must travel to and an operation that it must perform there.
- An optional *deadline*, indicating when the transport order is supposed to have been processed.
- An optional *type*, which is a string used for filtering vehicles that may be assigned to the transport order. A vehicle may only be assigned to a transport order if the order's type is in the vehicle's set of allowed order types. (Examples for potentially useful types are `"Transport"` and `"Maintenance"`.)
- An optional *intended vehicle*, telling the dispatcher to assign the transport order to the specified vehicle instead of selecting one automatically.
- An optional set of *dependencies*, i.e. references to other transport orders that need to be processed before the transport order. Dependencies are transitive, meaning that if order A depends on order B and order B depends on order C, C must be processed first, then B, then A. As a result, dependencies are a means to impose an order on sets of transport orders. (They do not, however, implicitly require all the transport orders to be processed by the same vehicle. This can optionally be achieved by also setting the *intended vehicle* attribute of the transport orders.) The following image shows an example of dependencies between multiple transport orders:

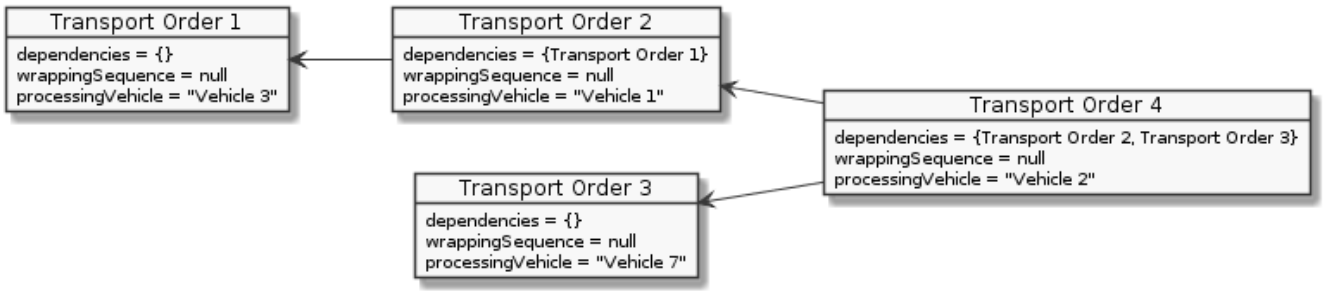

Figure 2. Transport order dependencies

2.3.2. Order sequence

The plant overview application currently does not provide a way to create order sequences. They can only be created programmatically, using dedicated clients that are not part of the openTCS distribution.

An order sequence describes a process spanning multiple transport orders which are to be executed subsequently — in the exact order defined by the sequence — by a single vehicle. Once a

vehicle is assigned to an order sequence, it may not process transport orders not belonging to the sequence, until the latter is finished.

Order sequences are useful when a complex process to be executed by one and the same vehicle cannot be mapped to a single transport order. This can be the case, for instance, when the details of some steps in the process become known only after processing previous steps.

An order sequence carries the following attributes:

- A sequence of *transport orders*, which may be extended as long the complete flag (see below) is not set, yet.
- A *complete* flag, indicating that no further transport orders will be added to the sequence. This cannot be reset.
- A *failure fatal* flag, indicating that, if one transport order in the sequence fails, all orders following it should immediately be considered as failed, too.
- A *finished* flag, indicating that the order sequence has been processed (and the vehicle is not bound to it, anymore). An order sequence can only be marked as finished if it has been marked as complete before.
- An optional *type* — see Transport order. An order sequence and all transport orders it contains (must) share the same type.
- An optional *intended vehicle*, telling the dispatcher to assign the order sequence to the specified vehicle instead of selecting one automatically. If set, all transport orders added to the order sequence must carry the same intended vehicle value.

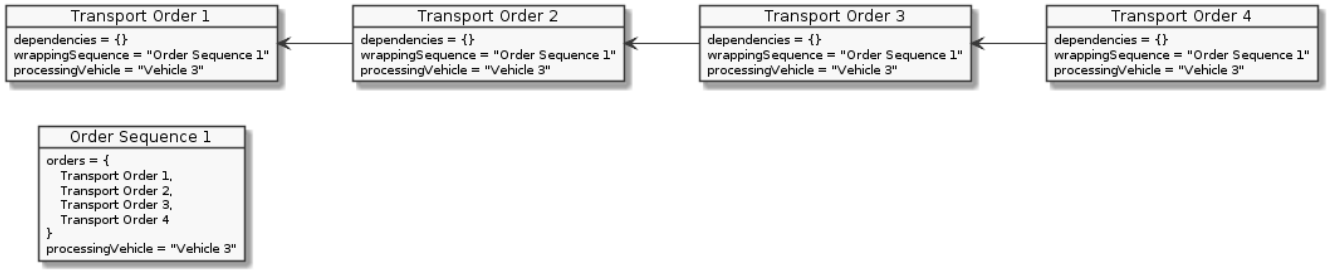

Figure 3. An order sequence

2.4. Common element attributes

2.4.1. Unique name

Every plant model and plant operation element has a unique name identifying it in the system, regardless of what type of element it is. Two elements may not be given the same name, even if e.g. one is a point and the other one is a transport order.

2.4.2. Generic properties

In addition to the listed attributes, it is possible to define arbitrary properties 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 adjusting the Plant Overview's configuration entry `initialMode` to `OPERATING` or `MODELLING` you can automatically start in the specific mode (see Plant Overview configuration).

3.1.1. Starting in modelling mode

1. Start the plant overview client (`startPlantOverview.bat/.sh`)

   By default it is configured to start in '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

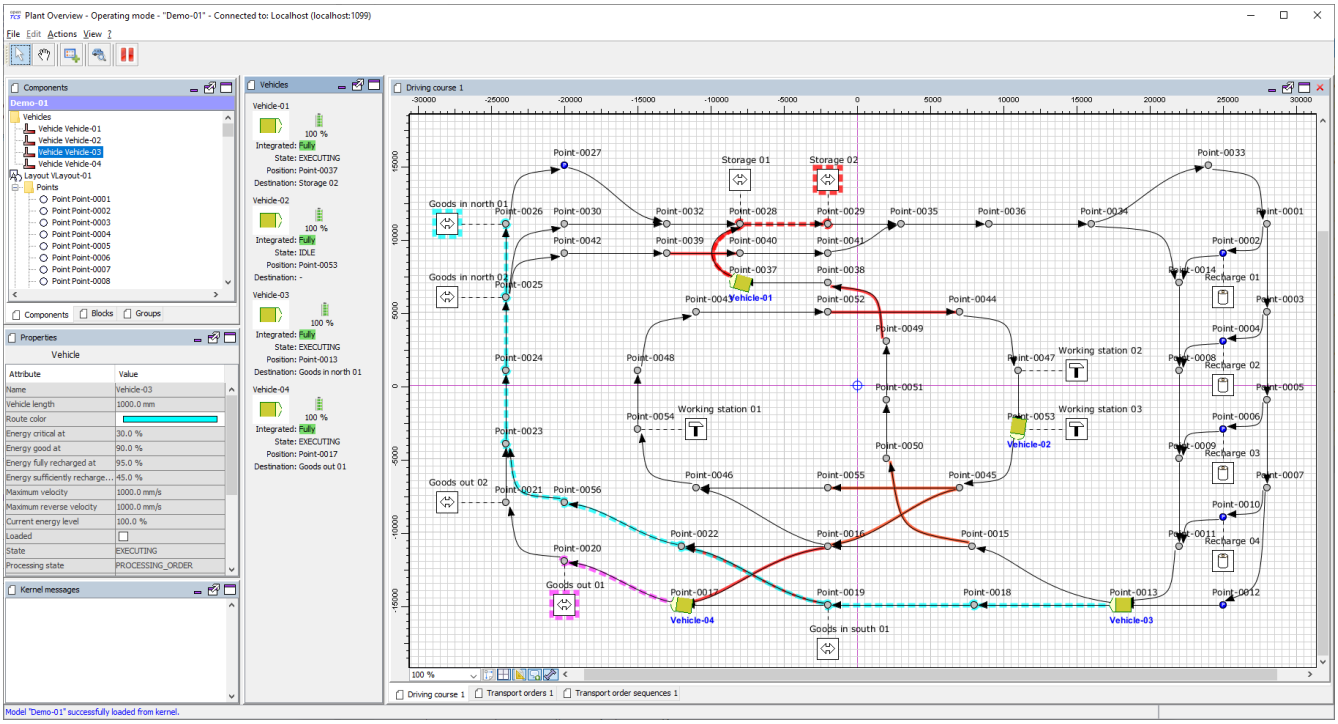

Figure 4. Plant overview client displaying plant model

1. Start the kernel (`startKernel.bat/.sh`).

a. If this is your first time running the kernel, you need to persist the current plant model first. Select **[File | Persist model in the kernel]** in the plant overview (also see Saving the plant model).

2. Optionally, start the kernel control center client (`startKernelControlCenter.bat/.sh`)
3. Start the plant overview client (`startPlantOverview.bat/.sh`)
   a. Switch the plant overview client to 'Operating mode' (**[File | Mode | Operating mode]**).
4. Select the tab **[Vehicle driver]** 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 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.

For now, all steps regarding the tab **[Vehicle driver]** can either be done in the kernel control center provided by the kernel itself or in the standalone/remote version. The kernel control center GUI in the kernel will be removed in openTCS 5.0, though.

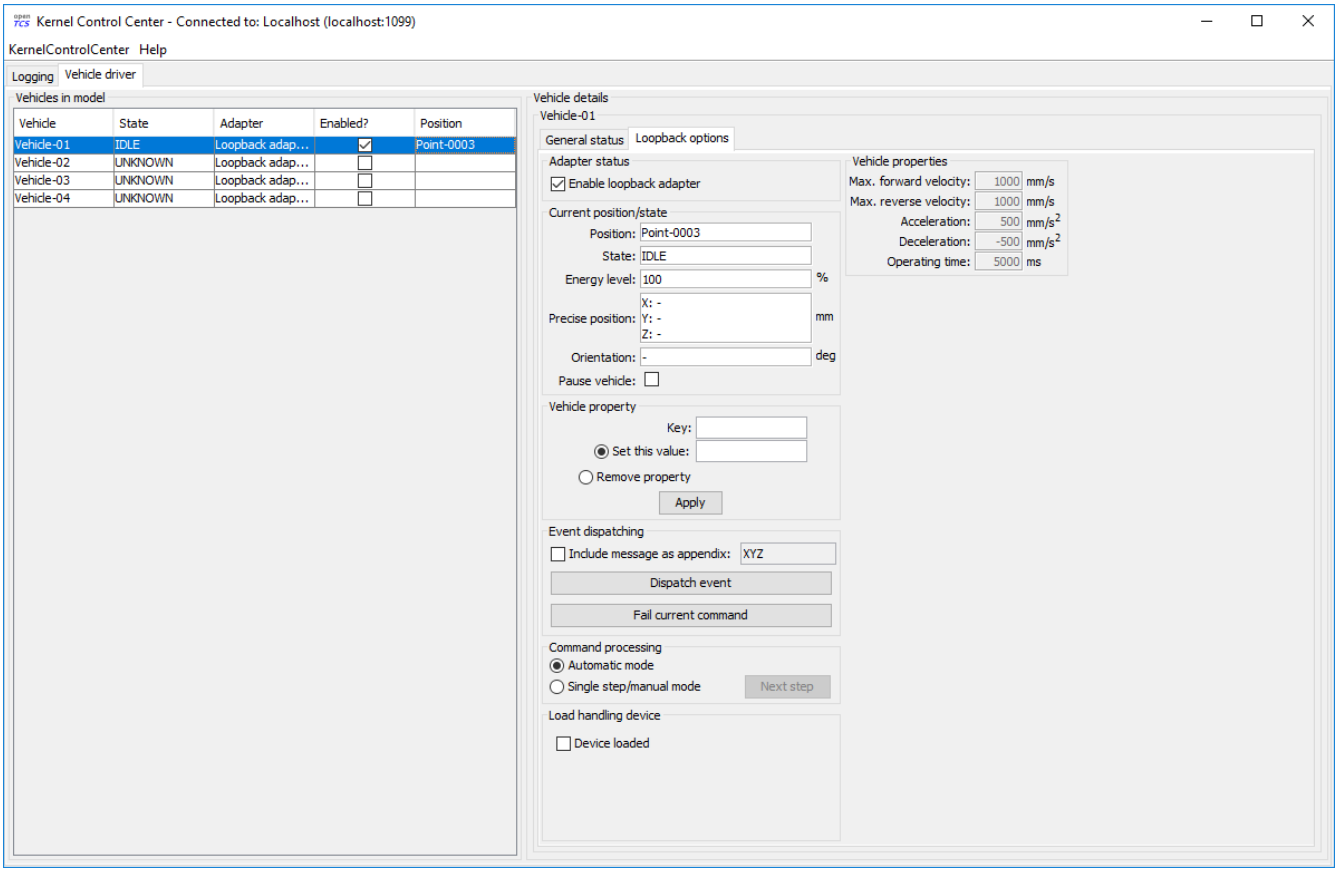

Figure 5. Driver panel with detailed view of a vehicle

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

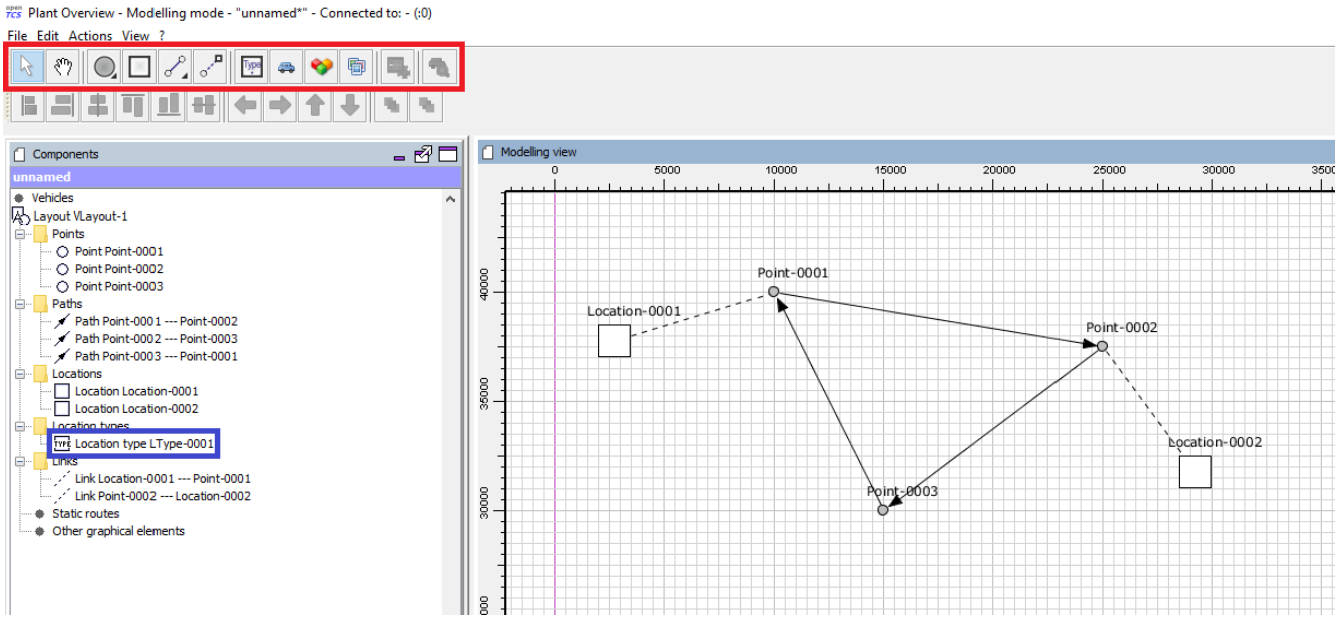

Figure 6. Control elements in the plant overview client (modelling mode)

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 plant

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. Optionally, start the kernel control center client (`startKernelControlCenter.bat/.sh`).
3. Start the plant overview client (`startPlantOverview.bat/.sh`), wait until its graphical user interface is shown and switch it to 'Operating mode'.

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. 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.
8. Right-click on the vehicle and select **[Context menu | Change integration level | ...to utilize this vehicle for transport orders]** to allow the kernel to dispatch the vehicle. The vehicle is then available for processing orders, which is indicated by an integration level `TO_BE_UTILIZED` 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 | Change integration level | ...to respect this vehicle's position]** in the context menu. The integration level shown is now `TO_BE_RESPECTED` and the vehicle will not be dispatched for transport orders any more.)

For now, all steps above can either be done in the kernel control center provided by the kernel itself or in the standalone/remote version. The kernel control center GUI in the kernel will be removed in openTCS 5.0, though.

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. You may also optionally select or define a type for the transport order to be created. 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. When withdrawing a transport order, its processing will be cancelled and the vehicle (driver) will not receive any further movement commands for it. A withdrawal can be issued by right-clicking on the respective vehicle in the plant overview client, selecting **[Context menu | Withdraw transport order]** and then selecting one of the following actions:

- '...and let the vehicle finish movement': The vehicle will process any movement commands it has already received and will stop after processing them. This type of withdrawal is what should normally be used for withdrawing a transport order from a vehicle.
- '...and stop the vehicle immediately': In addition to what happens in the case of a "normal" withdrawal, the vehicle is also asked to discard all movement commands it has already received. (This *should* make it come to a halt very soon in most cases. However, if and how far exactly it will still move highly depends on the vehicle's type, its current situation and how communication between openTCS and this type of vehicle works.) Furthermore, all reservations for resources on the withdrawn route (i.e. the next paths and points) except for the vehicle's currently reported position are cancelled, making these resources available to other vehicles. This "immediate" withdrawal should be used with great care and usually only when the vehicle is currently *not moving*!

Since an "immediate" withdrawal frees paths and points previously reserved for the vehicle, it is possible that other vehicles acquire and use these resources themselves right after the withdrawal. At the same time, if the vehicle was moving when the withdrawal was issued, it may - depending on its type - not have come to a halt, yet, and still move along the route it had previously been ordered to follow. As the latter movement is not coordinated by openTCS, this can result in a *collision or deadlock* between the vehicles! For this reason, it is highly recommended to issue an "immediate" withdrawal only if it is required for some reason, and only if the vehicle has already come to a halt on a position in the driving course or if other vehicles need not be taken into account. In all other cases, the "normal" withdrawal should be used.

Processing of a withdrawn transport order *cannot* be resumed later. To resume a transportation process that was interrupted by withdrawing a transport order, you need to create a new transport order, which may, of course, contain the same destinations as the withdrawn one. Note, however, that the new transport order may not be created with the same name. The reason for this is:

a. Names of transport orders need to be unique.
b. Withdrawing a transport order only aborts its processing, but does not remove it from the kernel's memory, yet. The transport order data is kept as historical information for a while before it is completely removed. (For how long the old order is kept depends on the kernel application's configuration — see Order pool configuration entries.)

As a result, a name used for a transport order may eventually be reused, but only after the actual data of the old order has been removed.

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** | **Change integration level** | **...to ignore this vehicle**] to disable the vehicle for transport order processing and to free the point in the driving course that the vehicle is occupying.
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.

For now, all steps regarding the kernel control center can either be done in the kernel control center provided by the kernel itself or in the standalone/remote version. The kernel control center GUI in the kernel will be removed in openTCS 5.0, though.

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 either a transport order or a vehicle becomes available, the dispatcher needs to decide what should happen with which transport order and which vehicle should do what. To make this decision, the default dispatcher takes the following steps:

1. New transport orders are prepared for processing. This includes checking general routability and unfinished dependencies.
2. Updates of processes that are currently active are performed. This includes:
    - Withdrawals of transport orders
    - Successful completion of transport orders
    - Assignment of subsequent transport orders for vehicles that are processing order sequences
3. Vehicles that are currently unoccupied are assigned to processable transport orders, if possible.
    - Criteria for a vehicle to be taken into account are:
        - It must be at a known position in the driving course.
        - It may not be assigned to a transport order, or the assigned transport order must be *dispensable*. That is the case with parking orders, for instance, or with recharging orders if the vehicle's energy level is not critical.
        - Its energy level must not be critical.
    - Criteria for a transport order to be taken into account are:
        - It must be generally dispatchable.
        - It must not be part of an order sequence that is already being processed by a vehicle.
    - The assignment mechanics are as following:
        - If there are less unoccupied vehicles than processable transport orders, the list of vehicles is sorted by configurable criteria. The default dispatcher then iterates over the sorted list and, for every vehicle, finds all orders processable by it, computes the required routes, sorts the candidates by configurable criteria and assigns the first one.
        - If there are less processable transport orders than unocuppied vehicles, the list of transport orders is sorted by configurable criteria. The default dispatcher then iterates over the sorted list and, for every transport order, finds all vehicles that could process it, computes the required routes, sorts the candidates by configurable criteria and assigns the first one.
        - For configuration options regarding the sorting criteria, see Default dispatcher configuration entries.

4. Vehicles that are still unoccupied are sent to a recharging location, if possible.
    - Criteria for a vehicle to be taken into account are:
        - It must be at a known position in the driving course.
        - Its energy level is *degraded*.

5. Vehicles that are still unoccupied are sent to a parking position, if possible.
    - Criteria for a vehicle to be taken into account are:
        - It must be at a known position in the driving course.
        - It must not be at a parking position already.

4.1.1. Default parking position selection

When sending a vehicle to a 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`.

4.1.2. Optional parking position priorities

Optionally (see Default dispatcher configuration entries for how to enable it), parking positions may be explicitly prioritized, and vehicles can be reparked in a kind of "parking position queues". This can be desirable e.g. to park vehicles close to locations that are frequent first destinations for transport orders. (For example, imagine a plant in which goods are transported from A to B all the time. Even if there currently aren't any transport orders, it might nevertheless be a good idea to prefer parking positions near A to reduce reaction times when transport orders arrive.)

To assign a priority to a parking position, set a property with the key `tcs:parkingPositionPriority` on the point. The property's value should be a decimal integer, with lower values resulting in a higher priority for the parking position.

4.1.3. Default recharging location selection

When sending a vehicle to a recharge location, 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: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.

4.2.1. Cost functions

The cost function used for evaluating the paths in the driving course can be selected via configuration — see Default router configuration entries. The following cost functions/configuration options are available:

- `DISTANCE`: Routing costs are equal to the paths' lengths.
- `TRAVELTIME`: Routing costs are computed as the expected time to travel on the paths, i.e. as path length divided by maximum allowed vehicle speed.
- `EXPLICIT`: Routing costs are read from the paths' *routing cost* attributes.
- `EXPLICIT_PROPERTIES`: Routing costs for a vehicle on a path are taken from path properties with keys `tcs:routingCostForward<GROUP>` and `tcs:routingCostReverse<GROUP>`. The `<GROUP>` to be used is the vehicle's routing group (see below). As an example, if a vehicle's routing group is "Example", routing costs for this vehicle would be taken from path properties with keys `tcs:routingCostForwardExample` and `tcs:routingCostReverseExample`. This way, different routing costs can be assigned to a path, e.g. for different types of vehicles.
  Note that, for this cost function to work properly, the values of the routing cost properties should be decimal integers.

The default cost function for a path simply evaluates to the path's length, so the cheapest route by default is the shortest one.

4.2.2. Routing groups

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 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 arbitrary string. (Vehicles that have the same value set share the same routing table, and the empty string 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), as described below.

4.3.1. Allocating resources

When an allocation of a set of resources for a vehicle is requested, the scheduler performs the following checks to determine whether the allocation can be granted immediately:

1. Check if the requested resources are generally available for the vehicle.
2. Check if the requested resources are part of a block with the type `SINGLE_VEHICLE_ONLY`. If not, skip this check. If yes, expand the requested resource set to the effective resource set and check if the expanded resources are available for the vehicle.
3. Check if the requested resources are part of a block with the type `SAME_DIRECTION_ONLY`. If not, skip this check. If yes, check if the direction in which the vehicle intends to traverse the block is the same the block is already being traversed by other vehicles.

If all checks succeed, the allocation is made. If any of the checks fail, the allocation is queued for later.

4.3.2. Freeing resources

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.

4.3.3. Fairness of scheduling

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. Application language

By default, all openTCS applications with user interfaces display texts in English language. The applications are prepared for internationalization, though, and can be configured to display texts in a different language, provided there is a translation for it. The openTCS distribution comes with the default (English) language and a German translation. Additional translations can be integrated — how this is done is described in the Developer's Guide.

For setting the language, each application has a configuration entry that needs to be set to a *language tag* for the language to use. (See Kernel Control Center application configuration entries and Plant Overview application configuration entries.) Examples for language tags are:

- "en" for English
- "de" for German
- "no" for Norwegian
- "zh" for Chinese

By default, the configuration entries are set to "en", resulting in English texts. Since a German translation is included, you can switch e.g. the Plant Overview to German by setting its `locale` configuration entry to "de". (Note that the application needs to be restarted for this.)

Configuring an application to use a language for which there is no translation will result in the default (English) language to be used.

5.2. Kernel configuration

The kernel application reads its configuration data from the following files:

1. `config/opentcs-kernel-defaults-baseline.properties`,
2. `config/opentcs-kernel-defaults-custom.properties` and
3. `config/opentcs-kernel.properties`.

The files are read in this order, and configuration values set in one file can be overridden in any subsequent one. For users, it is recommended to leave the first two files untouched and set overriding values and project-specific configuration data in `opentcs-kernel.properties` only.

5.2.1. 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
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.

5.2.2. 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)
locale	String	The application's current language, as a BCP 47 language tag. Examples: 'en', 'de', 'zh'
loggingAreaCapacity	Integer	The maximum number of characters in the logging text area.

5.2.3. 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).

5.2.4. Default dispatcher configuration entries

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

Table 4. Configuration options with prefix 'defaultdispatcher'

Key	Type	Description
orderCandidatePriorities	Comma-separated list of strings	Keys by which to prioritize transport order candidates for assignment. Possible values: BY_AGE: Sort by transport order age, oldest first. BY_DEADLINE: Sort by transport order deadline, most urgent first. DEADLINE_AT_RISK_FIRST: Sort orders with deadlines at risk first. BY_COMPLETE_ROUTING_COSTS: Sort by complete routing costs, lowest first. BY_INITIAL_ROUTING_COSTS: Sort by routing costs for the first destination. BY_ORDER_NAME: Sort by transport order name, lexicographically.
orderPriorities	Comma-separated list of strings	Keys by which to prioritize transport orders for assignment. Possible values: BY_AGE: Sort by age, oldest first. BY_DEADLINE: Sort by deadline, most urgent first. DEADLINE_AT_RISK_FIRST: Sort orders with deadlines at risk first. BY_NAME: Sort by name, lexicographically.
vehicleCandidatePriorities	Comma-separated list of strings	Keys by which to prioritize vehicle candidates for assignment. Possible values: BY_ENERGY_LEVEL: Sort by energy level of the vehicle, highest first. IDLE_FIRST: Sort vehicles with state IDLE first. BY_COMPLETE_ROUTING_COSTS: Sort by complete routing costs, lowest first. BY_INITIAL_ROUTING_COSTS: Sort by routing costs for the first destination. BY_VEHICLE_NAME: Sort by vehicle name, lexicographically.
vehiclePriorities	Comma-separated list of strings	Keys by which to prioritize vehicles for assignment. Possible values: BY_ENERGY_LEVEL: Sort by energy level, highest first. IDLE_FIRST: Sort vehicles with state IDLE first. BY_NAME: Sort by name, lexicographically.
deadlineAtRiskPeriod	Integer	The time window (in ms) before its deadline in which an order becomes urgent.
assignRedundantOrders	Boolean	Whether orders to the current position with no operation should be assigned.

Key	Type	Description
dismissUnroutableTransportOrders	Boolean	Whether unroutable incoming transport orders should be marked as UNROUTABLE.
rerouteTrigger	String	What triggers rerouting of vehicles. Possible values: NONE: Rerouting is disabled. DRIVE_ORDER_FINISHED: Vehicles get rerouted as soon as they finish a drive order. TOPOLOGY_CHANGE: Vehicles get rerouted immediately on topology changes.
reroutingImpossibleStrategy	String	The strategy to use when rerouting of a vehicle results in no route at all. The vehicle then continues to use the previous route in the configured way. Possible values: IGNORE_PATH_LOCKS: Stick to the previous route, ignoring path locks. PAUSE_IMMEDIATELY: Do not send further orders to the vehicle; wait for another rerouting opportunity. PAUSE_AT_PATH_LOCK: Send further orders to the vehicle only until it reaches a locked path; then wait for another rerouting opportunity.
parkIdleVehicles	Boolean	Whether to automatically create parking orders for idle vehicles.
considerParkingPositionPriorities	Boolean	Whether to consider parking position priorities when creating parking orders.
reparkVehiclesToHigherPriorityPositions	Boolean	Whether to repark vehicles to parking positions with higher priorities.
rechargeIdleVehicles	Boolean	Whether to automatically create recharge orders for idle vehicles.
keepRechargingUntilFullyCharged	Boolean	Whether vehicles must be recharged until they are fully charged. If false, vehicle must only be recharged until sufficiently charged.
idleVehicleRedispatchingInterval	Integer	The interval between redispatching of vehicles.

5.2.5. Default router configuration entries

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

Table 5. 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 6. Configuration options with prefix 'defaultrouter.shortestpath'

Key	Type	Description
algorithm	String	The routing algorithm to be used. Valid values: 'DIJKSTRA': Routes are computed using Dijkstra's algorithm. 'BELLMAN_FORD': Routes are computed using the Bellman-Ford algorithm. 'FLOYD_WARSHALL': Routes are computed using the Floyd-Warshall algorithm.
edgeEvaluators	Comma-separated 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. 'EXPLICIT': A route's cost is the sum of the explicitly given costs of its paths. 'EXPLICIT_PROPERTIES': Like 'EXPLICIT', but the costs are extracted from path properties.

5.2.6. Admin web API configuration entries

The kernel's admin web API can be configured using the following configuration entries:

Table 7. Configuration options with prefix 'adminwebapi'

Key	Type	Description
enable	Boolean	Whether to enable the admin interface.
bindAddress	IP address	Address to which to bind the HTTP server, e.g. 0.0.0.0. (Default: 127.0.0.1.)
bindPort	Integer	Port to which to bind the HTTP server.

5.2.7. Service web API configuration entries

The kernel's service web API can be configured using the following configuration entries:

Table 8. Configuration options with prefix 'servicewebapi'

Key	Type	Description
enable	Boolean	Whether to enable the interface.
bindAddress	IP address	Address to which to bind the HTTP server, e.g. 0.0.0.0 or 127.0.0.1.
bindPort	Integer	Port to which to bind the HTTP server.
accessKey	String	Key allowing access to the API.
statusEventsCapacity	Integer	Maximum number of status events to be kept.
useSsl	Boolean	Whether to use SSL to encrypt connections.

5.2.8. RMI kernel interface configuration entries

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

Table 9. Configuration options with prefix 'rmikernelinterface'

Key	Type	Description
enable	Boolean	Whether to enable the interface.
registryPort	Integer	The TCP port of the RMI.
remoteDispatcherServicePort	Integer	The TCP port of the remote dispatcher service.
useSsl	Boolean	Whether to use SSL to encrypt connections.
remoteKernelPort	Integer	The TCP port of the remote kernel.
remoteKernelServicePortalPort	Integer	The TCP port of the remote kernel service portal.
remotePlantModelServicePort	Integer	The TCP port of the remote plant model service.
remoteTransportOrderServicePort	Integer	The TCP port of the remote transport order service.
remoteVehicleServicePort	Integer	The TCP port of the remote vehicle service.
remoteNotificationServicePort	Integer	The TCP port of the remote notification service.
remoteSchedulerServicePort	Integer	The TCP port of the remote scheduler service.
remoteRouterServicePort	Integer	The TCP port of the remote router service.
clientSweepInterval	Long	The interval for cleaning out inactive clients (in ms).

5.2.9. XML host interface configuration entries

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

Table 10. Configuration options with prefix 'xmlhostinterface'

Key	Type	Description
enable	Boolean	Whether to enable the XML host interface.
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.

5.2.10. SSL server-side encryption configuration entries

The kernel's SSL encryption can be configured using the following configuration entries:

Table 11. Configuration options with prefix 'ssl'

Key	Type	Description
keystoreFile	String	The file url of the keystore.
keystorePassword	String	The password for the keystore.
truststoreFile	String	The file url of the truststore.
truststorePassword	String	The password for the truststore.

5.2.11. Statistics collector configuration entries

The kernel's statistics collector can be configured using the following configuration entries:

Table 12. Configuration options with prefix 'statisticscollector'

Key	Type	Description
enable	Boolean	Whether to enable the statistics collector.

5.2.12. Virtual vehicle configuration entries

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

Table 13. Configuration options with prefix 'virtualvehicle'

Key	Type	Description
enable	Boolean	Whether to enable to register/enable the loopback driver.

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.

5.3. Kernel Control Center configuration

The kernel control center application reads its configuration data from the following files:

1. `config/opentcs-kernelcontrolcenter-defaults-baseline.properties`,
2. `config/opentcs-kernelcontrolcenter-defaults-custom.properties` and
3. `config/opentcs-kernelcontrolcenter.properties`.

The files are read in this order, and configuration values set in one file can be overridden in any subsequent one. For users, it is recommended to leave the first two files untouched and set overriding values and project-specific configuration data in `opentcs-kernelcontrolcenter.properties` only.

5.3.1. Kernel Control Center application configuration entries

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

Table 14. Configuration options with prefix 'kernelcontrolcenter'

Key	Type	Description
locale	String	The kernel control center application's locale, as a BCP 47 language tag. Examples: 'en', 'de', 'zh'
connectionBookmarks	Comma-separated list of <description>|<hostname>|<port>	Kernel connection bookmarks to be used.
connectAutomaticallyOnStartup	Boolean	Whether to automatically connect to the kernel on startup. If 'true', the first connection bookmark will be used for the initial connection attempt. If 'false', a dialog will be shown to enter connection parameters.
loggingAreaCapacity	Integer	The maximum number of characters in the logging text area.

5.3.2. SSL KCC-side application configuration entries

The kernel control center application's SSL connections can be configured using the following configuration entries:

Table 15. Configuration options with prefix 'ssl'

Key	Type	Description
enable	Boolean	Whether to use SSL to encrypt RMI connections to the kernel.
truststoreFile	String	The path to the SSL truststore.
truststorePassword	String	The password for the SSL truststore.

5.4. Plant Overview configuration

The plant overview application reads its configuration data from the following files:

1. `config/opentcs-plantoverview-defaults-baseline.properties`,
2. `config/opentcs-plantoverview-defaults-custom.properties`,
3. `config/opentcs-plantoverview.properties`.

The files are read in this order, and configuration values set in one file can be overridden in any subsequent one. For users, it is recommended to leave the first two files untouched and set overriding values and project-specific configuration data in `opentcs-plantoverview.properties` only.

5.4.1. Plant Overview application configuration entries

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

Table 16. Configuration options with prefix 'plantoverviewapp'

Key	Type	Description
locale	String	The plant overview application's locale, as a BCP 47 language tag. Examples: 'en', 'de', 'zh'
initialMode	String	The plant overview application's mode on startup. Valid values: 'MODELLING', 'OPERATING'
frameMaximized	Boolean	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.

Key	Type	Description
frameBoundsY	Integer	The GUI window's configured y-coordinate on screen in pixels.
connectionBookmarks	Comma-separated list of <description>|<hostname>|<port>	Kernel connection bookmarks to be used.
useBookmarksWhenConnecting	Boolean	Whether to use the configured bookmarks when connecting to the kernel. If 'true', the first connection bookmark will be used for the connection attempt. If 'false', a dialog will be shown to enter connection parameters.
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.

5.4.2. SSL PO-side application configuration entries

The plant overview application's SSL connections can be configured using the following configuration entries:

Table 17. Configuration options with prefix 'ssl'

Key	Type	Description
enable	Boolean	Whether to use SSL to encrypt RMI connections to the kernel.
truststoreFile	String	The path to the SSL truststore.
truststorePassword	String	The password for the SSL truststore.

5.4.3. Plant Overview element naming scheme configuration entries

The plant overview application's element naming schemes can be configured using the following configuration entries:

Table 18. Configuration options with prefix 'elementnamingscheme'

Key	Type	Description
pointPrefix	String	The default prefix for a new point element.
pointNumberPattern	String	The numbering pattern for a new point element.
pathPrefix	String	The default prefix for a new path element.
pathNumberPattern	String	The numbering pattern for a new path element.
locationTypePrefix	String	The default prefix for a new location type element.
locationTypeNumberPattern	String	The numbering pattern for a new location type element.
locationPrefix	String	The default prefix for a new location element.
locationNumberPattern	String	The numbering pattern for a new location element.
linkPrefix	String	The default prefix for a new link element.
linkNumberPattern	String	The numbering pattern for a new link element.
blockPrefix	String	The default prefix for a new block.
blockNumberPattern	String	The numbering pattern for a new block.
groupPrefix	String	The default prefix for a new group.
groupNumberPattern	String	The numbering pattern for a new group.
layoutPrefix	String	The default prefix for a new layout element.
layoutNumberPattern	String	The numbering pattern for a new layout element.
vehiclePrefix	String	The default prefix for a new vehicle.
vehicleNumberPattern	String	The numbering pattern for a new vehicle.

Chapter 6. Advanced usage examples

6.1. Configuring automatic startup

1. 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. Configuring a virtual vehicle's characteristics

The loopback driver supports some (limited) configuration of the virtual vehicle's characteristics via properties set in the plant model. You can set the properties the following way:

1. Start the plant overview client in modelling mode and create or load a plant model.
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 **[Miscellaneous]**. In the dialog shown, add a property key and value according to the list below.
4. Save the model and persist it in the kernel as described in Saving the plant model.

The loopback driver interprets properties with the following keys:

- `loopback:initialPosition`: Set the property value to the name of a point in the plant model. When started, the loopback adapter will set the virtual vehicle's current position to this. (Default value: not set)
- `loopback:acceleration`: Set the property value to a positive integer representing an acceleration in mm/s^2. The loopback adapter will simulate vehicle movement with the given acceleration. (Default value: 500)
- `loopback:deceleration`: Set the property value to a negative integer representing an acceleration in mm/s^2. The loopback adapter will simulate vehicle movement with the given deceleration. (Default value: -500)

- `loopback:loadOperation`: Set the property value to a string representing the virtual vehicle's load operation. When the virtual vehicle executes this operation, the loopback adapter will set the its load handling device's state to *full*. (Default value: not set)
- `loopback:unloadOperation`: Set the property value to a string representing the virtual vehicle's unload operation. When the virtual vehicle executes this operation, the loopback adapter will set its load handling device's state to *empty*. (Default value: not set)
- `loopback:operatingTime`: Set the property value to a positive integer representing the virtual vehicle's operating time in milliseconds. When the virtual vehicle executes an operation, the loopback adapter will simulate an operating time accordingly. (Default value: 5000)

6.4. Running kernel and its clients on separate systems

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

By default, both, the plant overview client and the kernel control center client are configured to connect to a kernel running on the same system. To connect them 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` and the kernel control center's application configuration parameter `kernelcontrolcenter.connectionBookmarks` to `SomeDescription|myhost.example.com|1099`. The configuration value can be a comma-separated list of <description>|<host>|<port> sets. The plant overview client and kernel control center client will automatically try to connect to the first host in the list. If that fails, they will show a dialog to select an entry or enter a different address.

6.5. Encrypting communication with the kernel

By default, client applications and the kernel communicate via plain Java Remote Method Invocation (RMI) calls or HTTP requests. These communication channels can optionally be encrypted via SSL/TLS. To achieve this, do the following:

1. Generate a keystore/truststore pair (`keystore.p12` and `truststore.p12`).
    a. You can use the Unix shell script or Windows batch file (`generateKeystores.sh/.bat`) provided in the kernel application's directory for this.
    b. The scripts use the key and certificate management tool 'keytool' that is included in both the Java JDK and JRE. If 'keytool' is not contained in the system's `Path` environment variable the `KEYTOOL_PATH` variable in the respective script needs to be modified to point to the location where the 'keytool' is located.
    c. By default, the generated files are placed in the kernel application's `config` directory.
2. Copy the `truststore.p12` file to the client application's (plant overview and/or kernel control center) `config` directory. Leave the file in the kernel application's `config` directory as well.
3. In the kernel's configuration file, enable SSL for the RMI interface and/or for the web service

interface. (See RMI kernel interface configuration entries and/or Service web API configuration entries for a description of the configuration entries.)

4. If you enabled SSL for the RMI interface, you need to enable it in the Plant Overview's and the Kernel Control Center's configuration files, too. (See SSL PO-side application configuration entries and SSL KCC-side application configuration entries for a description of the configuration entries.)

6.6. 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.

6.7. 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.shortestpath.edgeEvaluators` 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.
- `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, TRAVELTIME"`, costs for routes are computed as the sum of the paths' lengths and the time a vehicle needs to pass it. If none of these entries is set, costs for routes are computed by the paths' lengths by default (`DISTANCE`).

6.8. 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).

6.9. 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.