Course manual

Rave I

Version 22 of Crew Pairing, Crew Rostering and Tail Assignment

Göteborg Nov 2016

Table of contents

General

About this document

This document contains course overheads and exercises.

About Rave I course

The course gives you a complete and thorough understanding of the Rave language. After completing the course you will be able to:

- read and understand existing Rave code
- maintain existing rule sets
- implement new rule sets.

Rave

Rave is a global modeling language that describes legality, costs and quality constraints. Rave supports all Crew & Fleet products and is essential both for tuning everyday production and performing simulations. Rave is also used as a stand-alone product for in-house applications.

Rave allows modeling of complex rules and cost functions that can be checked quickly by optimizers. The performance is essential, since an optimization job may have to do billions of rule and cost evaluations.

Besides providing legality and cost functions for the optimizers, the definitions in Rave are also available for reports and GUI customization.

Slides

Exercises

Introduction

Variable names

Variable names are written in font courier: `variable_name`. Use the suggested variable names since they may be required in reports or in other exercises.

Useful levels

- Leg atomic object
- Duty at least 8 hours between arrival and departure
- Trip chain identifier

Log on

Step 1 Double click the Academy Desktop icon 

You are now logged in to windows as student## (see sticker on screen)

Step 2 Double click the Exceed on Demand icon 

Step 3 Log in to unix as: **student## password**

Step 4 Double click the Launcher start icon for the RAVEI_SYSTEM 
Just press **Yes/Run** in the certificate warning pop ups

Step 5 Enter your **password** again

Step 6 In the Launcher change Roles to **Developer** by right clicking in the background (if needed)

Step 7 Start Studio by clicking the Studio icon 

Since you have not done anything yet the system cannot find a rule set, and will produce a warning message: read, understand, and click **OK**.

Note It is always important to read all messages and make sure you understand them.

If the message is unreadable, try to increase the size of the message window.

Note In these exercises you will use a rule set where modules are used. This is the case for most clients but not all.

Rave programmer's tools

Exercise 1 Purpose

To learn the system's general functionality and special tools for developing rules.

Exercise 1.1 Getting started

Step 1 Load a sub-plan.

1. Select command **Planning Tools >Plan Manager** to load a sub-plan.
2. Open the `Course/Rave/RaveI/Exercises` sub-plan for this exercise. Since you still have not compiled and loaded a rule set, you will get a warning message about missing rules.
3. Select **Show Trips** in window 1.

> **Hint:** Press the blue rectangle with the number one in the top left part of Studio.

Step 2 Compile a rule set.

1. Select command **Admin Tools > Rule Source Manager…**
2. Select the rule set `RaveI_ruleset`
3. Click **Build…** to compile.
   Make sure that *only* the two check boxes **Studio** and **with RAVE Explorer** are selected.
4. Wait for the compilation to finish.
   You will be notified with a pop-up.

Step 3 Load a rule set.

1. Select command **AdminTools > Rule Set Manager**
2. Select directory `All rule sets` and file `RaveI_ruleset` and click **Load Rule Set**.
3. Verify in the window title that RaveI_ruleset is loaded.

Step 4 Look at all the dictionary variables containing the string `block` or the string `connect` in the rule set:

1. Right click on any leg.
   The **Trip Object** menu is displayed.
2. Select command **Rave > Show Rule Values…** in the menu.

Note If the Rave menu is not visible, try to use Refresh button (double green arrow)

3. Set **Variable**, **Module** to `*` and type search string `block|connect` in the **Name** field.
   Click **Ok**.

Note The **Show Rule Values** functionality may be used to make sure that new variables return proper values, and to investigate why certain variables have certain values. With the Show Rule Values form you may also see how the variables depend on other variables.

Step 5 Try using different values in the Show Rule Values form, for example set **Level** to `leg`.

> **Hint:** Read the online manual: *Planning* > *Studio Commands* > *Show Rule Values*

Step 6 Start the Developer Workspace:

1. Select command **Admin Tools: Developer Workspace.**
2. Select 2-3 legs in Studio, use the rubber-band technique.
3. In the Rule set view (to the right in DWS) find the `RaveI_ruleset > Studio > leg` module and double click to open in editor.
4. In the Outline view, use the sort A->Z button and find the `%start_station%` variable. Click to jump to that definition in the editor.
5. On the definition, **Right click > Add Watch** to add this variable to the Gantt Rave Explorer view at the bottom.
6. Verify that you get three columns, one for each selected leg in Studio.
7. Click on the arrows and double click on the variables in the Explorer view.

Step 7 Display variable values using Rave Explorer.

1. Right-click a leg and select command **Trip Object: Rave > Rave Explorer**.
   The Rave Explorer form is displayed.
2. Select **Module** `leg` and **Variable** `start_station`.
   When you press **OK**, the variable value, its dependencies and source references will be displayed in Rave IDE.

> **Hint:** You may need to press `Rave IDE Project` and `Rave IDE File Overview` the first time Rave IDE is opened.

3. For more information, see:
   Rave Manual > *Tools*, section *Rave IDE*.

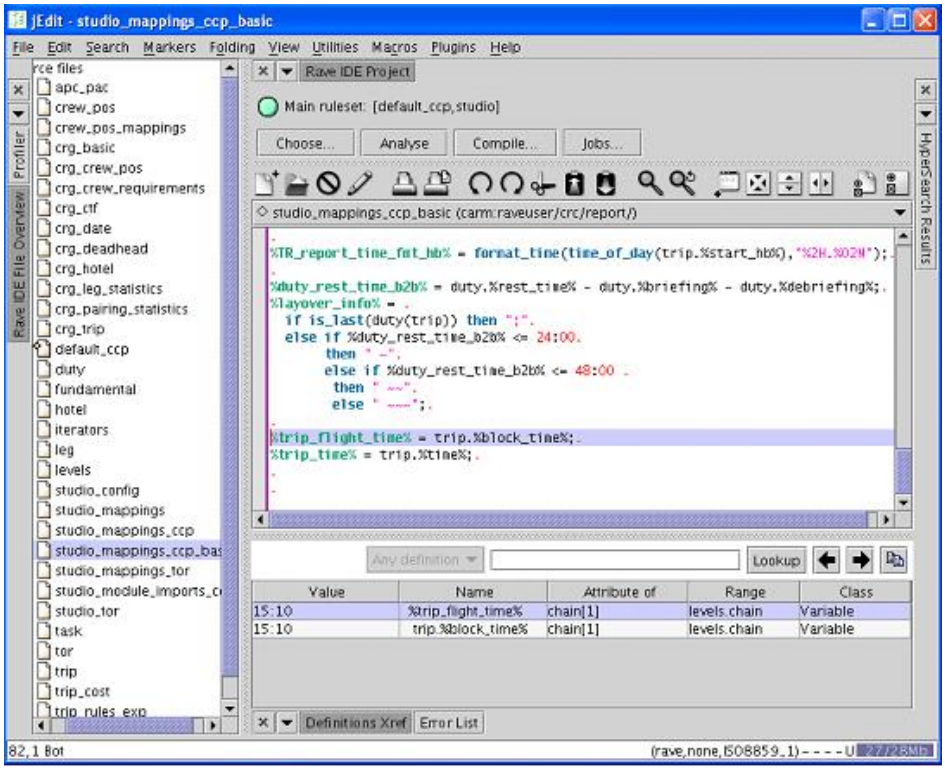


Figure 1 Rave IDE. Visible plug-ins are:
Rave IDE File Overview, Rave IDE Project, Definitions Xref

Step 8 Display Rave variables using the Rave evaluator.

1. Select command **Special: Rave > Rave Evaluator**
2. Write `leg.%start_station%` in the Rave Expression field.
3. Press Eval
4. Hover with the mouse (Mouse-Over) over different legs. The Rave evaluator will display the values of `leg.%start_station%` for the leg that you are currently hovering over.

Note The Special menu is part of the 'NiceToHave' package with (undocumented and unsupported) functionality that is useful for developers.

Exercise 1.2 Rule parameters

Step 9 Select **Window1**: **Show Legal/Illegal > Show Illegal Trips**.

Hint: Click on the blue rectangle with '1' in the top left part of Studio.

Step 10 Look at the parameters by selecting **Planning Tools: Rule Parameters** or click toolbar button

Step 11 Click **Basic Rules** tab in the Rule Parameters form and turn on the rule for **Min connection time for A/C change**.

Step 12 Change the **Min connection time** parameter to 1:15
Click **Ok**.

Step 13 Select **Window1**: **Show Legal/Illegal > Show Illegal Trips**.

Step 14 Select all legs in all trips with **Ctrl-A**.

Step 15 Right click on a leg and select the command **Trip Object: Check Legality** to get a rule failure report.

> **Hint:** Illegal rules are also shown in the Info window (bottom left) when you hover over the Trip header column (left hand side).

Step 16 Try to 'click' in the report.

Exercise 1.3 Select trips

Select the legs for which the connection is greater than 1:30

Step 17 Select **Trip General**: **Select > by …**
A selection form is displayed.

> **Hint:** Press right mouse button anywhere in the window background to access the General menu, or press key '2'.

Step 18 Select **Filter principle** `Any` and **Select** `Leg` so Studio will select legs that meet the criteria in the form.

Step 19 Look at the fields at the bottom of the form under **Rule Values** (you may have to scroll down in the form).

Step 20 Write `leg.connection_time` on the left-hand side and `>1:30` on the right-hand side. Then click **OK**.

Step 21 Also look at the different options at the top of the *filter* form (command **Trip General: Filter > by …**).

What does the filter method `SUBFILTER` do?
What do the other options mean?

> **Hint:** Read the online documentation *Planning> Crew Planning User Guide> Studio> Working in Studio> Selecting and deselecting objects* and *Filtering*

Data Types and Keywords

Exercise 2 Purpose

Try mixing data types and understanding keywords

Exercise 2.1 Mix data types

Step 1 Make sure you have 2-3 legs selected in Studio

Step 2 In DWS, open Gantt Rave Explorer view

Step 3 Add a new expressions (make sure to evaluate each expression before adding the next one), type:

1. `arrival`
2. `arrival-departure`
3. `arrival-departure/0:01 (How can this be fixed?)to fix this (arrival-departure)/0:01`
4. `arrival+True` **`it can't be`**
5. `arrival+0:07`

Step 4 Select different legs and click the refresh icon to re-evaluate.

Exercise 2.2 View the keywords list

Step 5 Show all available keywords. Select the command **Help: Keywords etc.** A new window/browser is opened displaying help on keywords, iterators, contexts and transforms.

Step 6 Click a keyword in the list to get a detailed description.

Step 7 How many keywords start with '`arrival`'? 9

Step 8 *In Studio, do Show Rule Value and type * in the keywords field.

Step 9 View available keywords with Rave IDE.
In Rave IDE, Definitions Xref: write `*` in the search field, select `Keywords` in the drop-down list and click **Lookup.**

Note Keywords, iterators, contexts, and transforms are all *application* dependant. Rave just supplies the possibility to define them; it does not know anything about them. This means that documentation is part of the Studio documentation.

Variables and Parameters

Exercise 3 Purpose

To understand variables and parameters.

Write these definitions in the **ravei_exercises** module and **Save** the file.
Then **compile** `RaveI_ruleset`, (re-) **load** this rule set
and evaluate the result of the variables to test them.

Hint: Use DWS for all work with Rave code.

Find RaveI_ruleset in the **Project Explorer** view > right click > properties
Rule Set
Select only Studio [Ok]
(Or click the 'Choose active rule set' button (R with Tick).)

Hint: To edit a file, find it in the **Ruleset** view (to the right) and double click on it

Hint: To compile the rule set, select the source file in the **Ruleset** view
Click on the compile button
Select and turn on the **Explorer** option and **Reload in Studio**

Exercise 3.1 Basic definitions, block time

Block time starts (aircraft departs) when blocks* are off, and ends (aircraft arrives) when blocks are on after a landing.

*) blocks are used to stop aircraft wheels

Define all of these new variables from scratch in the module called **ravei_exercises**

Step 1 Define block time for a single leg — `block_time`

Hint: Use suitable keywords.

Step 2 Define the variables briefing and debriefing as parameters. — `briefing` `debriefing`
Add the remarks: — `"My Briefing Time: "`
and — `"My Debriefing Time: "`

Step 3 Define start and end of the active planning period as parameters — `period_start`, `period_end`

Step 4 Define the start time for briefing before a leg — `briefing_start`

Step 5 Define the minimum allowed connection time to 0:45 — `min_cxn_time_p`

Exercise 3.2 Verify

Use DWS Rave Explorer to verify all new definitions.

Step 6 Select the Gantt Rave Explorer Tab at the bottom of DWS

Step 7 Find the variable in the Outline view to the right (or in editor), right-click and select the **Add watch** command

Step 8 Make sure you have one or a few legs selected in Studio

Step 9 If DWS is not connected to a Studio session: In the Rave Evaluator view, click on **No RPC Servers** to the left, and select your Studio session

Step 10 Click on the white **Arrow** icon next to the evaluated expression to see sub-expressions

Step 11 Click the refresh icon to re-evaluate all expressions
e.g. new selection of legs or new rules

Exercise 3.3 Parameter Form

Simple exercise to learn about the layout file for the parameter form.

Step 12 Open the parameter form and in tab [ALL PARAMETERS], search for your new definitions.

Step 13 Open the file: `crc/form/groupdefs_Developer` and uncomment the `ravei_exercises` group definition

Step 14 Open the parameter form again, and verify that your new tab [Exercises] is available.

Functions and Built-in Functions

Exercise 4 Purpose

Learning how to define functions and use Built-in functions.

Exercise 4.1 Functions

Step 1 Create a Boolean function that takes a reltime as an argument and returns True for times between 12:00 and 14:00

`is_during_lunch_time`

Exercise 4.2 Square root

Step 2 Is it possible to take the square root of a relative time?
Try sqrt(ravei_exercises.%block_time%)

Step 3 … what can be done? It has to be with declaration var, but also it has to be an integer

Exercise 4.3 Define Time of day

Step 4 Define the time of day for a leg start `leg_start_tod`

Exercise 4.4 Night duty

Step 5 Flight time between 22:00-6:00 counts as 20% extra time. For example, flying time between 00:00 and 05:00 counts as six hours work.

`flight_time_with_night_compensation`

Example: Flight is from 04:00 to 10:00.

%flight_time_with_night_compensation% =
(04:00 to 06:00)*1.2 + (6:00 to 10:00) = 6:24

Hint: Use scale_time.

if-then-else, Tables

Exercise 5 Purpose

To understand if-then-else and tables

Exercise 5.1 Block time

Step 1 Create a new block time variable that returns 0:00 if a leg is a deadhead otherwise it returns the air time `block_time_2`

Exercise 5.2 Leg points – internal tables

Step 2 Define a Boolean variable that identifies legs that start late, after 20:00 `is_late_start`

Step 3 Define leg points according to the table. `points_simple`

Block Time	Points (late start, >=20:00)	Points (start < 20:00)
< 1:00	1	1
1:00 – 1:30	3	2
1:30 – 2:00	3	3
2:00 – 2:30	5	4
>= 2:30	7	5

Note `%points_simple%` will be used later, make sure it is correct.

External Tables and Sets

Exercise 6 Purpose

To work with external data files

Exercise 6.1 External tables

Step 1 Use hotel costs from the external table `hotel_cost` (already exists in your system under the 'Plan Tables: Sub-plan tables directory) and calculate the cost of a layover depending on the layover country

`cost_of_layover`

Country	Cost of Layover
NL	1000
IT	500
CZ	800
DE	1100
FR	800
CH	900
All Others	999

The existing etable **SpLocal/hotel_cost.etab** has the following content:

```
2

Scountry "Country",
Icost_of_layover "Cost",
"CH",   900,
"CZ",    800,
"DE",    1100,
"FR",   800,
"IT",    500,
"NL",    1000,
```

Step 2 If there is no match in the table, 999 should be returned.

Step 3 Add new cost for missing countries to complete the table

Note Don't forget to refresh the table in Studio

Exercise 6.2 Set

Step 4 Create an external set with all countries from the hotel table where you can have a layover

`possible_layover_country_set`

Step 5 Verify if an arrival station is in the set

`is_allowed_layover_country`

Exercise 6.3 More sets

The minimum connection time depends on the arrival airport of the leg (keyword arrival_airport_name).

Minimum connection time for the leg should be 45 minutes for legs arriving to CDG or ZRH, 30 minutes for other locations.

Step 6 First make it possible to change the cities that require the longer connection time.

`long_connection_airports_p`

> **Hint:** You should define a set for this

Step 7 Create a function that returns minimum connection time for a certain airport. Do not use a table for this exercise.

`min_connection_time_for_airport(airport)`

Example:

```
%min_connection_time_for_airport%("GOT") = 0:30
```

Step 8 Create a variable that gives minimum required connection time for the current leg. Use the previously defined function to avoid duplication of code.

`min_connection_time`

Note `min_connection_time` will be used later, make sure it is correct.

Step 9 Turn all hard coded times into parameters.

Step 10 Add planner remarks to all your new parameters.

> **Hint:** In the on-line help, search for 'variables and functions' in the Development Book
> Click on 'Variables and functions' and then 'Remark'.

Step 11 Find your parameters in the parameters form – select one and press F1.

> **Hint:** You may have to restart Studio for this to work properly

Exercise 6.4 Additional Exercises, Tables:

In addition to `%points_simple%` crew members are also awarded extra points on weekends according to the following table.

Leg starts on weekday (points_simple)	Weekend leg
1	1
2	3
3	4
4	5
5	6
6	7
7	7

Step 12 Create a variable to identify weekend legs `is_on_weekend`

Step 13 Create a new table that includes the new criteria points

> **Hint:** Use the built-in function `time_of_week()`

Exercise 6.5 Additional exercise, External Tables:

Define a function `%num_hotels_with_costs%(Int cost)` that, given a price of a hotel, returns the number of hotels with that price (from `hotel_cost.etab`).

Example

2
Scountry "Country",
Icost_of_layover "Cost",

"CH", 900,
"CZ", 800,
"DE", 1100,
"FR", 800,
"IT", 500,
"NL", 1000,

%num_of_hotel_with_cost%(1000) = 1
%num_of_hotel_with_cost%(800) = 2

Step 14 Verify by selecting the command **Show Rule Values > Call Function**.

> **Hint:** Use the count() traverser within a table.
> See *Development Book > Rave Manual > Definitions > Tables Aggregation of values in result rows.*

Levels

Exercise 7 Purpose

Working with Levels and using filters instead of where conditions.

Note You will almost never have to work with levels in an existing user, but it is good to know the basics.

Note Levels impact performance, make sure they are really needed and fast to evaluate.

Exercise 7.1 Levels

Step 1 Create a new level, based on legs, which ends when you have enough time for a meal break (1:20) after the leg. Use `leg.%connection_time%`.

```
leg_is_last_in_meal_break
level meal_block
```

Verify by clicking on a trip and select the command **Rave > Show Rule Values**. Check that a new meal_block level starts after a break of at least 1:20. Select any keyword, e.g. `local_arrival_time_summer`

Show Rule Values:
Level: `ALL`
Keyword: `local_arrival_time_summer`

Step 2 Or verify by selecting legs that have enough time for meal break:

General > Select > by...
[Default]
Select > [Leg] (at the top)
<scroll down> to Rule Values
ravei_exercises.leg_is_last_in_meal_break True
[Ok]

Traversers, Void Values and Filters

Exercise 8 Purpose

Working with Traversers, shifting focus of current/active leg and handling Void values.

Exercise 8.1 Traversers

Step 1 Define total block time in a duty `duty_block_time`

Step 2 Define total points in a duty `duty_points`
(This variable will be used later)

Step 3 Define block time of the longest leg in a duty `max_leg_block_time`

Step 4 Define a variable which counts the number of legs in a duty `duty_num_legs`

Step 5 Define number of duties in a trip `trip_num_duties`

Step 6 Define number of legs in a trip `trip_num_legs`

Step 7 Define layover airport name `layover_airport_name`
(last arrival airport in the duty)

Step 8 Define departure time of the first leg in a trip `trip_dep`

Step 9 Define a variable which is true if there is any deadhead leg in a duty, otherwise false `has_deadhead`

Step 10 Define connection time as the difference of the departure of the next leg and the arrival of the current leg (this concerns legs in a duty) `cxn_time`

Exercise 8.2 Duty time definitions

Step 11 Define duty start as first departure in duty – briefing `duty_start`
and duty end as last arrival in duty + debriefing `duty_end`
Reuse the briefing and debriefing parameters from exercise 3.

Step 12 Define duty time in a duty. `duty_time`
(A duty starts at `%duty_start%` and ends at `%duty_end%`)

Step 13 Define total duty time in a trip. `trip_duty_time`
(the `sum` of all `%duty_time%`)

Step 14 Count accumulated block time inside a duty. Accumulated block time means total block time of all legs before and including the currently evaluated leg. Thus, this value will be different for each leg in a duty `acc_block_time`

Exercise 8.3 Duty in period

Step 15 Define duty time in current planning period for a duty.
Reuse the period parameters from exercise 3
`time_in_period`

Hint: Use overlap.

Exercise 8.4 Void

Step 16 With your current definition of `cxn_time` it will sometimes return void. Create a new variable where void is replaced by `8:00`.
`cxn_time2`

Exercise 8.5 Additional Exercises

Step 17 Define a variable that is True if this leg has the most block time of all legs in the trip.
`has_most_block_time`

Step 18 In a trip, find the sequence number of the last leg with the most block time.
`last_most_block`

Notice, there could be several legs that have same block time which is also max block time. “Take” the last leg.

Step 19 Find a better name to the previous variable.

Step 20 In addition to the night compensation, also count duty time during the weekend days as 20% extra. There should be no double compensation during the weekend nights.
(Weekend: Fri 22:00 - Mon 06:00. A duty is always shorter than a week.)
`time_with_night_and_weekend_compensation`

Leg_

Modules, DWS, Rave Profiler

Exercise 9 Purpose

To work with modules and variable definitions in multiple files, hands-on experience with DWS.

Exercise 9.1 Using modules

Step 1 Look in the existing modules and try to find duplicates of the variables from the previous exercises.

Hint: Look at: duty.%briefing%, trip.%duty_time%

Step 2 How is the start of a duty defined?
Why are there different ones?

Exercise 9.2 Global Export

Step 3 Look at the code in file 'fundamental'. What type of variables are there, and why are most of them 'globally' exported? Most of them are global export

Step 4 In file 'levels', all level definitions are globally exported; is it still possible to write e.g. levels.leg? – Try it!

Step 5 In file 'leg' implement a new variable that defines the minimum absolute time to 1jan2000

`min_abstime`

Step 6 What happens?

Exercise 9.3 Find, Search and Replace

Let's look more at the DWS.

Step 7 Examine the Edit menu

Step 8 Start: Find/Replace …

Step 9 Examine the Search menu
Look at File Search
and how to define a 'Working Set'

Exercise 9.4 Variable completion

Step 10 In any module, define a new variable that uses the previous one.
Type the first characters of that variable...
Also try Ctrl-Space

Step 11 You should now get a list of possible continuations for the name you are typing. You can move around in the list using the up/down arrows. Select the desired variable with [Enter]

Step 12 Also test Shift-Ctrl-L

Exercise 9.5 Rave Profiler

The Rave Profiler can be used to find out where Rave spends the most of the time; i.e. which variables take the longest to evaluate.

The Rave Profiler is available both in DWS and Rave IDE.

Step 13 Compile RaveI_ ruleset for Rave Profiler.

Step 14 Reload rule set in Studio

Step 15 Start profiler: Admin Tools>Rave Profiler>Start Rave Profiler

Step 16 Run some Studio applications that require Rave calculation. For example run reports trip.py, deadhead.py and hotel_statistics.py. You can also run the reports multiple times. Another test could be to show illegal trips multiple times (then all Rave rules need to be evaluated).

Step 17 Stop profiler: Admin Tools>Rave Profiler>Stop Rave Profiler

Step 18 Save profiler: Admin Tools>Rave Profiler>Save Profile As…

Step 19 In DWS, click on Rave Profiler perspective (Stopwatch)

Step 20 Select the profiler file and/or press Load

Step 21 In the *Rave IDE*, click on the Profiler tab in the left margin

Step 22 Make the frame larger by drag'n'drop-ing the dotted vertical bar

Step 23 Open the profile file by pressing [Load]

Step 24 The profiler now shows the result of running applications in Studio.

Step 25 Examine the graph, hover over the boxes and also try to double-click on a box and view the variable in the Rave Explorer.

Step 26 Increase the Threshold to 2 in the Toolbar

Rules

Exercise 10 Purpose

To understand rules.

Use the command (report) **Check Legality** to investigate illegalities.

Exercise 10.1 Maximum block time per duty

Define a maximum block time per duty rule to 6 hours.
Define the limit as a parameter. `max_duty_block_time`, `max_duty_block_time_p`

Exercise 10.2 No layovers at certain airports

Disallow layovers at the airports AMS and GVA (there is a layover between two duties). `no_layover_at_airports`

Hint: Use a set.

Exercise 10.3 Minimum connection time

Connection time between two flights may not be less than minimum required connection time.

Write the rule so that it is tested on the second leg in the pair that forms a connection. Re-use the limit from previous exercises. Note that `%min_cxn_time%` normally gives minimum connection time *after* the leg.

rule: `min_cxn_time`

Experiment with the parameters from the same exercise in order to create illegal trips.

Exercise 10.4 Rest time rule

Each duty must be followed by a rest period equal to or larger than the duty time. The required rest will increase with 1:30 for each duty point (`%duty_points%`). rule: `min_rest_time`
limit**:** `min_required_rest`

Exercise 10.5 Maximum tough duties

A duty is a tough duty if it exceeds 7:40 of duty time.

At most one of three consecutive duties may exceed 7:40 hours of duty time. Define a rule to regulate this.

help var: `duty_is_tough`
rule: `max_tough_duties`

Exercise 10.6 Sliding rule

It is not allowed to have more than 16:00 hours of duty time in any consecutive 48 hours.

`max_duty_time_in_48_hours`

Exercise 10.7 Additional exercises

Step 1 Rewrite the rule in Exercise 10.3 so that it 'looks' forward instead. Which version do you prefer, why?

Step 2 Limit the number of consecutive tough duties (see Exercise 10.5 above).

Let all values be parameters (number of allowed consecutive tough duties, duty time for a tough duty).

Replace the old rule with this new one.

Step 3 Adjust the rule `trip_rules_exp.min_connection_time` so that legs *departing* from the airport GVA require an additional connection time of 30 minutes when they *arrive*.

Costs

Exercise 11 Purpose

To understand cost.

Exercise 11.1 Layover Costs

Define a new cost element for all the layovers in one trip.

`cost_of_all_layovers`

Exercise 11.2 Deadhead cost

Define a cost for the deadheads in a trip. There should be a basic cost of 1000 if there is any deadhead and an additional cost of 10 for each deadhead minute.

`cost_of_all_deadheads`

Contexts and Iterators

Exercise 12 Purpose

To try out some report code.

Exercise 12.1 Add module

Preparation

Step 1 Create a file in the modules directory called `report_ravei`
(Remember the module statement at the top.)

Step 2 Make sure that this file is used by the file `RaveI_ruleset` in the source directory, and that the use statement is added inside a
`#if product (Studio)`
clause. By placing the use statement inside the 'if' clause, these definitions will be invisible to the optimisers.

Exercise 12.2 Statistics – using iterators

In your new module, write definitions for calculating:

Step 3 average duty time for all duties in the current context.
`avg_duty_time`

> **Hint:** Import and use the predefined iterators.duty_set

Step 4 number of duties with night duty in the current context. Use scale_time().
`is_night_duty,`
`num_night_duties`

Step 5 number of trips with any night duty in the current context.
`trips_with_night_duty`

Step 6 total cost of all the trips in the current context.
Use trip_cost. %trip_total_cost_apc%. `total_cost`

Step 7 Verify the result by using the report `RaveI_report_1.py`.
Show Trips, select all trips and run command **Trip Object**: **Generate Report...**.

> **Hint:** Context is a Rave feature that is not covered by this course. However, the exercises require some knowledge:
>
> A context is the set of objects you start with when you apply iterators. Contexts are defined by the planning application, and their contents may vary.
> It is possible to explicitly specify the context in Rave. This is not needed in these exercises since the current context is the default.
> When reports are generated, the current context consists of one single

chain, all chains in a window or all trips/rosters in the plan, depending on how the report is started, i.e. from where it is started.

With **Show Rule Values** the current context is the single chain.
With Rave Explorer the current context depends on how it is used:
- Right click an object: single chain
- Right click in background: chains in the window

You can also change the context by selecting 'Objects' in the form.

Exercise 12.3 Using reports

Step 8 Start report `RaveI_report_2.py` from window by selecting **Trip Object**: **Generate Report**...
There is an error; you will fix it in the next steps.

Exercise 12.4 Flights – iterators with matching criteria

Step 9 Define a flight number iterator (in the `iterators` module) and use it to calculate the number of different flight numbers in the current context. Do not count flights which are only used as deadheads.

```
flight_number_set
number_of_flight_numbers
```

Step 10 The report looks into the different flight number bags. Count the number of flights in such a bag.

```
number_of_flights
```

Step 11 Reload rules and run the report again.

Exercise 12.5 Additional exercise

In the following exercise you will have to run a report, modify it, re-run it again, modify it...

In DWS, navigate to your report in the Project Explorer view and double click on it:
`RaveI_system/lib/python/report_sources/crr_window_object/`*`your_report`*

After a report is changed, it has to be re-loaded into Studio in order to enforce the changes:
your_report > Reload in Studio > *your_session*.

As you already may have noticed, it is possible to run report directly from DWS by selecting 'Generate Report ...'.

Connections – iterators with matching criteria

Step 12 Find the most common connection time in the sub-plan. If there are many connection times with same frequency, select the longest.
`connection_time_set`, `most_frequent_cxn_time`

Step 13 Uncomment *only* the part with `#self.add(self.c1())` in the Exercise 12.4 report so see your answer (which should be 0:40). Remove the # sign to uncomment the code.

Step 14 What would you have to change / add to display the total number of most frequent connections? `num_most_frequent_cxn_time`

Step 15 Uncomment *only* the part with `#self.add(self.c2())` in the Exercise 12.4 report to see your answer (which should be 26).

Note It is bad practice to use definitions like these for costs and rules. The evaluation takes a long time and will usually not work for the optimizers anyway. They should therefore only be used when generating reports.

Never use contexts or iterators for normal Rule and Cost code.

Rudobs and information window

Exercise 13 Purpose

To get an idea of how you may use Rave to change the GUI.

Study

Read the section about map values for Rudobs in *Development* > *Rave Manual* > Appendix: *Carmen User Information* > *Map values for rudobs*.

For your information

The variables that constitute the interface to Studio must be placed in one of the `_topmodule` files (`report` and `require` directories) and be included using the `require` statement. The variables referenced by the map variables (interface variables) might however, be defined in modules.

Exercise 13.1 RUle Defined OBjectS

Step 1 Re-use your briefing variables from earlier exercises for the rudobs, search in `studio_mappings` to find where you can connect them.

Step 2 Look in the parameters form and change the lengths of your parameters. Does it affect the GUI?

Exercise 13.2 Information window

Show total duty points of a trip in the fourth line of the information window.

Step 3 To change the content in the information window, change the file `show_info_flight.py` in the hidden report source directory.

> **Hint:** DWS, Project Explorer View, the_Carmuser_project > lib > python > report_sources > hidden

Step 4 Find the definition of `flight_info_4` and replace the empty string (`""`) in the last part with the proper Rave variable for the points (as in similar definitions for the other lines).

Step 5 On this line, should we show duty points (for all duties in the trip) or total trip points?

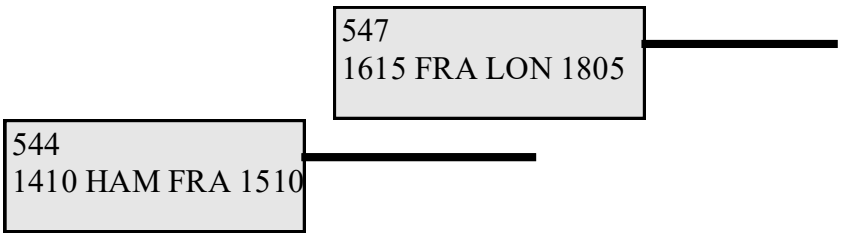
Step 6 Reload the report to make the new implementation work.

> **Hint:** Right click > Reload in Studio > newcastle:1234

Exercise 13.3 Additional Exercise, Create a Rudob

Step 7 Indicate, after each duty, the required rest time according to the earlier definition.

Example
Look at the file `studio_mappings` in the `report` directory. There are rudobs indicating briefing and debriefing (# 5 and 6). Define, in a similar way, a rudob indicating the required rest. You can use # 11.



Solutions

Exercise 1 Rave programmer's tools

It is important that you read and understand all error/warning messages.

Exercise 1.1 Getting started

There are many variables that have either `block` or `connection` in their names. You may restrict your search by matching start '`^`' and end '`$`' of a name, e.g. `^block_time$`.

Exercise 1.2 Select Trips

`SUBSELECT` will apply the filter criteria on the objects already in the window (`ADD` will add matching objects to the ones in the window).

Exercise 2 Data Types and Keywords

Exercise 2.1 Mix data types

arrival -> Abstime: 03Nov2003 08:00
arrival-departure -> Reltime: 1:35

Error, you may not mix Abstime and int data types,
dep/0:01 is evaluated first

(arrival-departure) / 0:01 -> Integer: 95

Error, you may not mix Abstime and Boolean data types

arrival + 0:07 -> Abstime: 03Nov2003 8:07

Exercise 2.2 View the keywords list

There are about 9 keywords that start with `arrival`.

Exercise 3 Variables and Parameters

Exercise 3.1 Basic definitions

```
%block_time% = arrival - departure;

%briefing% = parameter 0:45
    remark "My Briefing Time: ";

%debriefing% = parameter 0:30
    remark "My Debriefing Time: ";

%period_start% = parameter 01may97 00:00
    remark "Start of planning period";

%period_end% = parameter 30may97 00:00
    remark "End of planning period";

%briefing_start% =
    departure - %briefing%;

%min_cxn_time_p% = parameter 0:45
    remark "Minimum allowed connection time: ";
```

Exercise 4 Functions and Built-in functions

Exercise 4.1 Functions

```
%is_during_lunch_time%(Reltime a_time) =
    12:00 <= a_time
    and a_time <= 14:00;
```

Exercise 4.2 Square root

No, sqrt() only works on integers, but you may first divide the block time by 0:01.

Exercise 4.3 Time of day

```
%leg_start_tod% = time_of_day(departure);
```

Exercise 4.4 Night duty

```
%time_with_night_compensation% =
    scale_time(departure,arrival,10,22:00,6:00,12) / 10;
```

Exercise 5 if-then-else, Tables

Exercise 5.1 Block time

```
%block_time_2% =
    if deadhead then
```

```
        0:00
    else
        %block_time%;
```

Exercise 5.2 Leg points

```
%is_late_start% = %leg_start_tod% > 20:00;

table leg_points_tab =
    %block_time%,
    %is_late_start% -> %points_simple%;
    < 1:00,       False -> 1;
    (1:00, 1:30(, False -> 2;
    (1:30, 2:00(, False -> 3;
    (2:00, 2:30(, False -> 4;
    >=2:30,       False -> 5;
    < 1:00,       True -> 1;
    (1:00, 2:00(, True -> 3;
    (2:00, 2:30(, True -> 5;
    >=2:30,       True -> 7;
end
/* Note: some of the table rows could be combined
 * with a '-' instead of T/F */
```

Exercise 6 External Tables and Sets

Exercise 6.1 External tables

```
table hotel_cost_tab =
    arrival_airport_country -> int %hotel_cost%;
    external "SpLocal/hotel_cost.etab";
    country -> cost_of_layover;
    - -> void_int;
end
```

Use the Table Editor to add lines with some of the leg arrival countries that are missing. Use the 'refresh' button in Studio to consider your table changes.

Exercise 6.2 Set

```
set possible_layover_country_set =
    external String "SpLocal/hotel_cost.etab"."country";
%is_allowed_layover_country% =
    arrival_airport_country in possible_layover_country_set;
```

Exercise 6.3 More sets

```
set long_connection_airports_p = parameter "CDG", "ZRH"
    remark "Airports that require a long connection time: ";

%min_connection_time_for_airport%(String ap) =
    if ap in long_connection_airports_p then
        0:45
    else
        0:30;

%min_connection_time% =
    %min_connection_time_for_airport%(arrival_airport_name);

%long_min_connection_p% = parameter 0:45
    remark "Long min connection: ",
    planner "The minimum connection time for airports requiring
a long connection, see also %short_min_connection_p%";

%short_min_connection_p% = parameter 0:30
    remark "Short min connection: ",
    planner "The minimum required connection time for normal
airports, see also %long_min_connection_p%";
```

You should be able to find the parameters in the Exercises tab in the params form.

Exercise 6.4 Additional Exercise, Tables

```
%is_on_weekend% = 120:00 < time_of_week(departure);


table leg_points_tab_2 =
    %points_simple%, %is_on_weekend% -> %points%;
    (2,6), True -> %points_simple% + 1;
    -,- -> %points_simple%;

end
```

Exercise 6.5 Additional Exercise, External Tables

```
table num_hotels_tab(Int cost) =
    cost -> Int %num_hotels_with_cost%;
    external "SpLocal/hotel_cost.etab";
    cost_of_layover -> count(cost_of_layover);

end
```

Exercise 7 Levels

Exercise 7.1 Levels

```
level meal_block =
    is_last(leg)
    when(%leg_is_last_in_meal_block%);
end
```

```
%leg_is_last_in_meal_block% =
    leg.%connection_time%
    >= 1:20;
```

Exercise 8 Traversers, Void and Filters

Exercise 8.1 Traversers

```
%duty_block_time% = sum(leg(duty), %block_time%);
%duty_points% = sum(leg(duty), %points_simple%);
%max_leg_block_time% = max(leg(duty), %block_time%);
%duty_num_legs% = count(leg(duty));
%trip_num_duties% = count(duty(trip));
%trip_num_legs% = count(leg(trip));
                  /* sum(duty(trip), %duty_num_legs%) */
%layover_airport_name% =
    last(leg(duty), arrival_airport_name);
%trip_dep% = first(leg(trip), departure);
%has_deadhead% = any(leg(duty), deadhead);
%cxn_time% =
    next(leg(duty), departure) - arrival;
/* Will be void when there is no next leg in the duty*/
```

Exercise 8.2 Duty time definitions

```
%duty_start% = first(leg(duty),departure) - %briefing%;
%duty_end% = last(leg(duty),arrival) + %debriefing%;
%duty_time% = %duty_end% - %duty_start%;
%trip_duty_time% = sum(duty(trip), %duty_time%);
%acc_block_time% =
    sum(leg(duty), %block_time%)
    from(first)         /* Will be default, ie not needed */
    to(current);
```

Exercise 8.3 Duty in period

```
%time_in_period% =
    overlap(%duty_start%, %duty_end%,
            %period_start%, %period_end%);
```

Exercise 8.4 Void

```
%cxn_time2% = default(%cxn_time%, 8:00);
```

Exercise 8.5 Additional exercises

```
%has_most_block_time% =
    let this_bt = %leg_block_time%;
```

```
    this_bt = max(leg(trip), %leg_block_time%);

    /*%leg_block_time% = max(leg(trip), %leg_block_time%); also
works*/


%last_most_block% =
    last(leg(trip), %leg_num_in_trip%)
    where(%has_most_block_time%);
```

/* traverser `max` will also work */

```
%leg_num_in_trip% = count(leg(trip)) to (current);

%time_with_night_and_weekend_compensation% =
    let start = time_of_week(departure),
        stop = start + %block_time%;
        /* If the duty wraps Sun-Mon */


    /* duty time + (Sat/Sun 6-22)/5 (20% is 1/5)*/
    %time_with_night_compensation%
    + (overlap(start,stop,126:00,142:00)
       + overlap(start,stop,150:00,166:00))
       / 5;
```

Exercise 9 Modules, DWS, Rave Profiler

Exercise 9.1 Using modules

In a well-structured rule set it is simple to find already defined variables. Remember that you only write your code once, but it is read and maintained by all your colleagues forever – make their life easier.

```
%block_time% = leg.%block_time%;

%num_duties% = trip.%num_duties%;

%has_deadhead% = duty.%is_deadhead%;

%duty_start% =
       duty.%start_UTC%,
       duty.%start_lt%,
       duty.%start_hb%
```

Exercise 9.2 Global Export

The module `fundamental` mainly contains constants, help functions, and parameters that have no direct level dependency and could be used by any module. There should be no risk that other modules define other functions with the same name so to make it easier to use these functions they have all been globally exported.

Yes, it does. It works for normal variables because ‘global’ only means it is *possible* to omit the module name in other Rave modules. Remember that you always have to include the module name when writing reports or scripts.

On the other hand, you do not need to export variables if they are only used in scripts and reports.

You are not allowed to use the same variable name as globally exported variables form an imported module as Rave cannot determine which of the two you want to use.

Exercise 9.3 Find, Search and Replace

Exercise 9.4 Variable completion

`Ctrl-Space`, `Ctrl-Alt-/` are the shortcut key-combinations for variable completion.

Exercise 9.5 Rave Profiler

To activate the Rave Profiler functionality you have to compile for 'Profiler' using the Rave IDE. Then you can use Admin Tools : Rave Profiler … for Studio profiling. Optimization profiling will start automatically if the opt. ruleset is profiler compiled.

Exercise 10 Rules

Exercise 10.1 Maximum block time per duty

```
rule max_duty_block_time =
    %duty_block_time% <= %max_duty_block_time_p%;
    remark "Maximum block time per duty";
end

%max_duty_block_time_p% = parameter 8:00
    remark "Max block time per duty: ";
```

Exercise 10.2 No layovers at certain airports

```
set dissallowed_airports_set = parameter "AMS", "GVA"
    remark "No layover at these airports: ";

rule no_layover_at_airports =
    valid not is_last(duty(trip));
    not (%layover_airport_name% in dissallowed_airports_set);
    remark "No layover at certain airports";
end
```

Exercise 10.3 Minimum connection time

```
rule minimum_connection_time_bkw =
  valid not is_first(leg(duty));
```

```
  departure - prev(leg(duty), arrival)
  >= %prev_min_cxn_time%;
end
%prev_min_cxn_time% =
  prev(leg(duty), %min_connection_time%);
```

Exercise 10.4 Rest time rule

```
%rest_start% = %duty_end%;
%rest_end% = next(duty(trip), %duty_start%);
%rest_time% = %rest_end% - %rest_start%;
%min_required_rest% =
   %duty_time% + 1:30 * %duty_points%;
```

Note Remember, a rule where the expression is void is considered legal.

```
rule min_rest_time =
   valid not is_last(duty(trip));
   %rest_time% >= %min_required_rest%;
   remark "Minimum required rest time after a
duty";
end
```

Exercise 10.5 Maximum tough duties

```
%duty_is_tough% =
   if %duty_time% > 7:40
   then 1
   else 0;


rule max_tough_duties =
   %duty_is_tough%
   + default(prev(duty(trip), %duty_is_tough%), 0)
   + default(prev(duty(trip),
         prev(duty(trip), %duty_is_tough%)),0)
   <= 1;
   remark "Maximum one of three duties may exceed
7:40";
end
/* Same rule, but slightly faster execution (void
is legal) */
rule max_tough_duties_2 =
   valid %duty_is_tough% = 1;
   /* Now we know this duty is tough */
   prev(duty(trip),
         %duty_is_tough%
```

```
            + prev(duty(trip), %duty_is_tough%))

   < 1;
   remark "Maximum one of three duties may exceed
7:40";
end
```

Exercise 10.6 Sliding rule

```
rule max_duty_time_in_48_hours =
   %duty_time_in_48_hours%
   <= %max_duty_time_in_48_hours_p%;
   remark “Max duty time in 48 sliding hours”;
end
%max_duty_time_in_48_hours_p% = parameter 16:00
   remark "Maximum duty time in 48 consecutive
hours”;
%duty_time_in_48_hours% =
   let start = %duty_start%,
      stop = %duty_start% + 2*24:00;
   sum(duty(trip),
      overlap(%duty_start%, %duty_end%, start,
stop))
      from(current)
      while(%duty_start% < stop);
```

Exercise 10.7 Additional exercises

```
rule min_connection_time_fwd =
  valid not is_last(leg(duty));
  %cxn_time% >= %min_connection_time%;
  remark “Minimum required connection time (fwd):
”;
end
```

In this case it is probably easier to write the rule ‘forwards’ although it is the second leg that makes the connection illegal.

```
/* In file trip_rules_exp,
 *  compare with rule EXP_leg_min_connection_time
 */
%duty_is_tough2% =
   %duty_time% >= %tough_duty_limit_p%;
%tough_duty_limit_p% =
   parameter 7:40
   remark “Lower limit for when a duty is ‘tough’:
”;
```

```
%max_cons_tough_duties_p% =
   parameter 3
   remark "Max number of consecutive tough duties";
/* Count duties from the current duty forwards
   while %duty_is_tough% is true.*/
%cons_tough_duties% =
   count(duty(trip))
      from (current)
      while(%duty_is_tough2%);
rule max_cons_tough_duties =
   %cons_tough_duties%
   <= %max_cons_tough_duties_p%;
   remark "Max consecutive nr of tough duties";
end
```

In module trip_rules_exp

First change names of `%min_connection_time%` to `%_min_connection_time%`.

Then define:

```
%min_connection_time% =
   let buffer = if departure_airport_name = "GVA"
            then 0:30
            else 0:00;
   %_min_connection_time% + buffer;
```

Exercise 11 Costs

Exercise 11.1 Cost functions – external tables

```
/* Cost of layovers */
%layover_country% =
   last(leg(duty), arrival_airport_country);
table cost_of_layover_tab =
   %layover_country% -> Int %cost_of_layover%;
   External "hotel_cost";
   country -> cost_of_layover;
   - -> 1000;
end
%of_all_layovers% =
   sum(duty(trip), %cost_of_layover%)
   where( not is_last(duty(trip)) );
```

Exercise 11.2 Deadhead Cost

```
%cost_of_deadheads_basic_p% = parameter 1000
   remark "Cost of deadhead basic: ";
%cost_of_deadheads_minute_p% = parameter 10
   remark "Cost of each deadhead minute: ";
%cost_of_deadhead_minutes% =
   if deadhead then
      (arrival-departure) *
%cost_of_deadheads_minute_p%/0:01
   else
      0;
%cost_of_all_deadheads% =
   let basic = if any(leg(trip), deadhead) then
                  %cost_of_deadheads_basic_p%
               else
                  0;
   if basic > 0 then
      basic + sum(leg(trip),
                  %cost_of_deadhead_minutes%)
   else
      0;
```

Exercise 12 Statistics and reports

Exercise 12.1 Add module

In new file report_ravei

```
module report_ravei
/*
 * Empty file
 */
/* End of file */
```

In top source file

```
#if product(Studio)
   use report_ravei;
end
```

Exercise 12.2 Statistics – using iterators

In module report_ravei

```
import levels;
import iterators;
import trip_cost;
import ravei_exercises;
```

```
%avg_duty_time% =
   avg(duty_set, ravei_exercises.%duty_time%);
/* Remember to 'export' needed variables from prev.
ex:s */
%num_night_duties% =
   count(duty_set)
   where (%is_night_duty%);
%is_night_duty% =
   scale_time(ravei_exercises.%duty_start%,
              ravei_exercises.%duty_end%, 0,
             22:00, 6:00, 1)
   <> 0:00;
/* Overlap does not really work when the interval
 * crosses midnight like this.
 */
%trips_with_night_duty% =
   count(trip_set)
   where(any(duty(trip), %is_night_duty%));
/* Consider to define %trip_has_night_duty% */
%total_cost% =
   sum(trip_set,
      trip_cost.%trip_total_cost_apc%);
```

Exercise 12.3 Using reports

When you start the report generation from the Object menu then the context is the selected chains, this is the normal way to initiate reports.

Exercise 12.4 Flights- iterators with matching criteria

```
export iterator flight_number_set =
   /* group flights together */
   partition(leg)
   by (flight_number, deadhead);
end
/* The attribute deadhead is added so we can
 * filter flights only used as deadheads */


%number_of_flight_numbers% =
   count(iterators.flight_number_set)
   where (not deadhead);
%number_of_flights% = count(leg_set);
```

Verify that the report is correct.

Exercise 12.5 Additional exercise

Connections – iterators with matching criteria

In module iterators

```
export iterator connection_time_set =
   partition(leg)
   by(ravei_exercises.%cxn_time%);
end
```

In module report_ravei

```
%most_frequent_cxn_time% =
   let max_size =
      max(iterators.connection_time_set,
         count(leg_set));
   max(iterators.connection_time_set,
                    ravei_exercises.%cxn_time%)
   where(count(leg_set) = max_size);


%num_most_frequent_cxn_time% =
   max(connection_time_set,
      count(leg_set));
```

Exercise 13 Rudobs and information window

Exercise 13.1 RUle Defined OBjectS

Uncomment and comment out appropriate (already prepared lines) in `studio_config` to use your new parameters for the GUI rudobs.

Exercise 13.2 Information window

```
flight_info_4 = p.Text(' … FLT %5s DUT %5s DAYS %-5s Points
%s' % (
%ctx_bag.trip.block_time(),
ctx_bag.trip.duty_time(),
str(ctx_bag.trip.days()) or "-  ",
ctx_bag.ravei_exercises.points()
))
```

Duty point information should normally be put on the third row with the other duty values. The fourth row is for Trip values. You could define a new `%trip_points%` variable.

Exercise 13.3 Additional Exercise, Create a rudob

in report/studio_mappings

```
/* Length of the rudob */
%map_rudob_len_11_crr% =
   "studio_config.rudob_rest_len";


/* Start position of the rudob */
%map_rudob_start_11_crr% =
   "studio_config.rudob_rest_start";


/* Colour of rudob */
%map_rudob_color_11_crr% =
   "studio_config.rudob_rest_color";


/* Bottom position in % of leg height */
%map_rudob_bottom_11_crr% =
   "studio_config.rudob_rest_bottom";


/* Top position in % of leg height */
%map_rudob_top_11_crr% =
   "studio_config.rudob_rest_top";


/* Text shown when pointing on it*/
%map_rudob_text_11_crr% =
   "studio_config.rudob_rest_text";


/* Bar at the ends (true/false) */
%map_rudob_delimitation_11_crr% =
   "studio_config.rudob_rest_delimitation";
```

in module studio_config

```
import levels;
import ravei_exercises;
%display_rest_time% = parameter true
   remark "Display required rest time rudob";
%rudob_rest_start% = ravei_exercises.%duty_end%;
%rudob_rest_len% =
   if %display_rest_time%
      and is_last(leg(duty))
   then
      ravei_exercises.%min_required_rest%
   else
      0:00;
%rudob_rest_top% = parameter 50;
%rudob_rest_bottom% = parameter 45;
%rudob_rest_color% = "Red";
```

```
%rudob_rest_text% =
   format_time(%rudob_rest_len%,
               "Required rest after duty: %h:%02m");
%rudob_rest_delimitation% = true;
```

Quick reference

Constants

```
%briefing% = 0:45;
```

Parameters

```
%briefing% = parameter 0:45
   remark "Length of briefing: ";
```

Variables

```
%block_time% = arrival - departure;
```

Functions

```
%work_start_with_offset% (Reltime offset) =
   %start_time% + offset;
```

Built-in functions

```
%required_rest_after_leg% =
   nmax(%block_time%, %min_rest_allowed%);
%trip_number_of_days% =
   (round_up(%trip_end%, 24:00)
   - round_down(%trip_start%, 24:00)) / 24:00 ;
%rest_time_after_duty% =
   default(next(duty(trip), %duty_start%)
         - %duty_end%,
      0:00 );
```

Traversers

```
%trip_block_time% = sum(leg(trip), %block_time%)
%total_time_away% =
   last(leg (trip), arrival)
   - first(leg (trip), departure);
%accumulated_block_time% =
   let this_dep = departure;
```

```
    sum(leg(trip), %block_time%)
    while(departure <= this_dep);
```

If-Then-Else

```
%time_between_legs% =
    if is_last(leg(trip)) then
       0:00
    else
       next(leg(trip), departure) - arrival;
```

Tables

```
table hotel_costs_tab =
    %hotel% -> %hotel_cost%;
    "jerrys_inn" -> 250;
    "holliday_inn" -> 350;
    "sheraton" -> 500;
    "plaza" -> 650;
    - -> void_int;
end
```

External tables

```
table aircraft_family =
    aircraft_type -> String %aircraft_family%;
    external "aircraft_family";
    ac_type -> ac_family;
    - -> "no family";
end
```

External data file:

```
2
Sac_type,
Sac_family,
"747", "747",
"74E", "747",
"727", "727",
```

Set

```
set asian_airports = "BKK","SIN","HKG","PEK","NRT"
    remark "Airports in Asia: ";
%asian_landing% = arrival_airport_name in
asian_airports;
```

External Set

```
set ac_family_set =
  external "aircraft_family"."ac_family";
```

Filters

```
filter active_legs = leg(not deadhead);
```

Rule

```
rule min_rest_time_in_trip =
   %trip_rest_time% >= %min_rest_time_in_trip_p%;
   remark "Minimum rest in trip: ";
end
```

Levels

```
level trip =
   is_last( duty )
   when( %duty_arrival% = homebase );
end
```

Iterators

```
iterator flight_set =
   partition(leg)
   by(departure_airport_name, deadhead);
end
```

Context

```
context(sp_crrs, count(trip_set));
```

Transforms

```
transform(active_crew_chain_leg,
   transform(leg_ref,
      sum(leg_set, %tot_crew_leg%)));
```