```
// Dialogic-code for Railways Journey Planner (this English version code is not actually used,
// but translated from Dutch just in order to make it understandable for the non-Dutch reader)

version 0.95
language nl-NL    // default language
gendiagrams false // debug: regex diagrams, just for checking
//permissions ['NAME', 'DEVICE_PRECISE_LOCATION']
permission_required false

// Multi-language strings (actually not used in this version)
$confirm =
  nl-NL: "Ik verstond {0}"
  en-GB: "I heard {0}"
$askArrivalStation =
  nl-NL: "Waar wilt u heen? Noem een station|Noem aankomststation"
  en-GB: "To which railway station would you like to travel?"
  de-DE: "Zum welchen Bahnhof möchten Sie reisen?"
$geenStation =
  nl-NL: "Ik verstond {0}. Ik herken dit niet als stationsnaam"
  en-GB: "I heard {0}, but do not recognize this as a railway station name"
$explanationAboutStations =
  nl-NL: "Geef de naam van een station op"
  en-GB: "Enter the name of the railway station"
$bevestigAankomstStation =
  nl-NL: "U wilt naar {0}"
  en-GB: "You want to go to {0}"
$ConfirmStationSelection =
  nl-NL: "U wilt van {0} naar {1}"
  en-GB: "You will travel from {0} to {1}"
$askDepartureStation =
  nl-NL: "Vanaf welk station vertrekt u naar {0}?|Noem vertrekstation"
  en-GB: "From which railway station do you go to {0}|?"
$bevestigVertrekStation =
  nl-NL: "U vertrekt vanaf {0}"
  en-GB: "You will depart from {0}"

synonyms
  (a an)
  (leave depart)


dialog Reisplanner
```

```
(
  out string [] Dstat, // departure station, 2 entries: full name + technical abbreviation
  out string [] Astat,
  out DateTime TravelDate,
  out DateTime TravelMoment,
  out bool D_or_A,
  out string TravelMomentDA,
  out bool FreeSubscription,
  out bool InclReservationRequired,
  out bool AdditionalOptions,
  out bool NeedExtraTransferTime,
  out int ExtraTransferMinutes
)

customcode CustomCode

signatures // methods implemented in custom code. Templates are generated automatically
  string [] CheckStation (string [] station)
  void DeterminePossibleJourneys (string dStat, string aStat, string via, Date journeyMoment, bool d_or_a,
    int xtraTransferMinutes, bool inclResrvationRequired, bool freeSubscription, bool transferOK,
    out int nrOfAlternatives, out int curAlternative, out string [,] prices)
  void ObtainJourneyDetails (int index, out string dTime, out string aTime, out string platformNo,
    out int nrOfTrainChanges, out string interchangeStations, out string journeyTime,
    out string journeySchedHtml, out string journeySchedText, out string interchangeHtml, out string
interchangeText)

mongo_url_loc
  "mongodb://localhost/reisplanner"
mongo_url_dev
  "mongodb+srv://${mongoUsr}:${mongoPwd}@dereisplanner-jmlem.gcp.mongodb.net/test"
mongo_url_pro
  `mongodb://${mongoUsr}:${mongoPwd}@dereisplanner-shard-00-00-jmlem.gcp.mongodb.net:27017,
   dereisplanner-shard-00-01-jmlem.gcp.mongodb.net:27017,
   dereisplanner-shard-00-02-jmlem.gcp.mongodb.net:27017/
test?ssl=true&replicaSet=DePeisplanner-shard-0&authSource=admin&retryWrites=true`

activities travel_advice, additional_options

{
  // global variables
  Date travelDateTime = <today>;
  string timeString = "";
```

```
    string dateString = "";
    string dayName;
    int hhh = -1;
    int mmm = -1;
    int dayPart;
    bool started = false;
    bool reported = false;
    bool user_done = false;
    bool get_journey_plan = false;
    bool do_check_early = false;
    string journeySchedHtml = ""; // browser html
    string journeySchedText = ""; // smart device screen text
    string interchangeHtml = ""; // browser html
    string interchangeText = ""; // smart device screen text
    string journeyTime = "";
    int nrOfTrainChanges = 0;
    string interchangeStations = null;
    int nrOfAlternatives = 0;  // number of journey alternatives found
    int curAlternative = -1;   // index of selected journey
    int optRmhNo;        // index of optimal journey
    string [,] prices = null;
    string copyright = 'U+00A9';
    //string backArrow = '<<';
    //string forwArrow = '>>;
    //string pricesIco = 'U+20ACU+20AC' // double euro sign
    Date timetableExpiryDate = MakeDate (2019, 12, 8);

     // Triggers -- a trigger is a piece of code that is executed by the Dialog Driver
     //             each time its precondition is (reset from false to) true

    trigger Initialize () precondition (!started)
    {
      [!nl-NL!] // set language to Dutch (sort of obsolete now, dynamic switching is not possible. For browser only)
      started = true;
      enable activity travel_advice exclusive ; // multiple activities can be active at the same time
      say ('Hello this is your journey planner|Journey Planner') // => converted to SimpleResponse speech|text
    }


    trigger Finalize () precondition (user_done)
    activity travel_advice
    {
```

```
    card ("Possible journey", null, journeySchedText, null, null); // BasicCard
    halt("Thank you, I wish you a pleasant journey");
  }


  // Questions -- a question is executed when both its argument is
  //              'unknown' and the possible precondition is true. It contains patterns + code

  question AskArrivalStation (Astat) before AskDepartureStation // 'Astat' is only asked for when its
  // value (and that of Question parameters in general) is unknown. Such values can be in the following
  // states: unknown (unk), set, i.e. they have a value (set), or irrelevant (irv). These states can be
  // explicitly assigned. Only 'unk' values will be asked for, and then their state will change to 'set'.
  activity travel_advice
  ask $askArrivalStation
  explain $explanationAboutStations // explanation text is spoken after wrong answer only
  {
    [ DateTime date = <today> ] // script code is in [ square brackets ]
    ( ( i want to ( travel | go ) ) to !! // !! acts like a Prolog cut, preventing backtracking
      StationName (out Astat) // StationName is a Pattern, see below.
    | ( i ( want | have to ) travel
        ( on? Date(out date, out dateString, out dayName)
          [ TravelDate = date ] // variable value settings are undone upon failure / backtracking
        )?
        ( Moment (date == <today>, out TravelMoment, out timeString) // <today> etc. are predefined values
        )?
        ( from StationName (out Dstat) )?
      )
      ( ( ( travel | leave | go )? to )
        StationName (out Astat)
        [ if (is_set (TravelMoment)) D_or_A = true ] // assumption: time supplied is departure time
      | arrive ( at station? |in )
        StationName (out Astat)
      )
    )
    [ if (is_unk (TravelDate) && is_set (TravelMoment)) // assumption: today
        TravelDate = <today>
    ]
  }


  question AskDepartureStation (Dstat)
  activity travel_advice
```

```
    ask ($askDepartureStation, Astat [0])
    //explain $explanationAboutStations
    {
      ( [?nl-NL?]
        ((ik wil?)? ( vertrekken | vertrek))? vanaf
      | [?en-GB?]
        ((i ( want to)?)? ( leave | depart))? from
      )?
      StationName (out Dstat)
    }


    trigger ConfirmStationSelection () precondition (any_set (Dstat, Astat))
    {
      if (all_set (Dstat, Astat))
      {
        if (Dstat [1] == Astat [1])
        {
          say ("The departure station is equal to the arrival station");
          say ("Please provide both station names again");
          set_unk (Dstat, Astat) // => set to unknown, so they will be asked for again
        }
        //else
        //  say ($ConfirmStationSelection, Dstat [0], Astat [0])
      }
      else if (is_set (Dstat))
        say ($confirm, Dstat [0])
      else if (is_set (Astat))
        say ($confirm, Astat [0])
    }


    question AskWhen (TravelDate) after AskDepartureStation
    activity travel_advice
    ask ("When would you like to travel?|")
    suggest('now', 'tomorrow', 'day after tomorrow')   // @N => N milliseconds <break> (seconds if N < 10)
    explain "Say: now @600 tomorrow @600 friday @600 or next monay @800 of provide a date"
    {
      [ TravelDate = <today> ]  // assumption
      ( Moment (true, out TravelMoment, out timeString)
        [ D_or_A = true ]
      | ( (i want to travel)? on | (ik wil reizen)? op )?
        Date (out TravelDate, out dateString, out dayName)
        ( Moment (TravelDate == <today>, out TravelMoment, out timeString)
          [ D_or_A = true ]
        )?
      )
    }


    trigger CheckDateMoment0 () before CheckDateMoment1 precondition (all_set (TravelDate, TravelMoment, D_or_A))
    activity travel_advice
    {
      if (hhh <= 6) do_check_early = true;
    }


    trigger CheckDateMoment1 () precondition (!do_check_early && all_set (TravelDate, TravelMoment, D_or_A))
    activity travel_advice
    {
      set_unk (TravelMomentDA);

      if (TravelDate > timetableExpiryDate)
      {
        say (`The current timetable expires on {0}'|Timetable expires on {0}`, timetableExpiryDate);
        // text before | is spoken, after | on display. If | at end, display text = speech text
        set_unk (TravelDate, TravelMoment);
      }
      else
      {
        travelDateTime = MakeDateTime (TravelDate, TravelMoment);
        timeString = TimeToString (travelDateTime);

        if (TravelDate == <today>)
          dateString = "today";
        else
          dateString = Format ("on {0} {1}", DayName (TravelDate), DateToString (TravelDate));

        TravelMomentDA = Format ("{0} at {1}", (D_or_A ? "Departure" : "Arrival"), timeString);
      }
    }


    question ConfirmMomentPriorTo6am () precondition (do_check_early)
    activity travel_advice
    ask (`Do you mean {0} in the morning of {0} in the afternoon? Say morning or afternoon|
         {0} in the morning of {0} in the afternoon?`, timeString)
    suggest('morning', 'afternoon')
    {
      <any>* // zero or more arbitrary words may precede
      ( ( morning | ochtend | s? ochtends )
        [ say ("Ok, in the morning") ]
      | ( afternoon | middag | s? smiddags )
        [ TravelMoment = MakeDateTime (TravelDate, hhh+12, mmm);
          timeString = TimeToString (TravelMoment);
          say ("Ok, in the afternoon")
        ]
      )
      [ do_check_early = false ]
    }


    question AskIfMomentIsDepartureOrArrivalTime (D_or_A) after AskWhen
    activity travel_advice
    ask ("{0}", is_set (TravelMoment)
      ? Format ("Is {0} a departure time or an arrival time?|Departure time or an arrival time?", TimeToString
  (TravelMoment))
      : "Do you want to provide a departure time or an arrival time?|Departure time or arrival time?")
    suggest('Departure time', 'Arrival time')
    {
      <any>*
      ( departure time? [D_or_A = true]
      | arrival time? [D_or_A = false]
      )
    }

    question AskWhatTime (TravelMoment) after AskIfMomentIsDepartureOrArrivalTime
    activity travel_advice
    ask ("What time do you want to {0}?", D_or_A ? "leave" : "arrive")
    suggest((D_or_A && TravelDate == <today>) ? ['now', 'within half an hour', 'within an hour'] : [])
    {
      Moment (TravelDate == <today>, out TravelMoment, out timeString)
    }


    question AskAdditionalOptions (AdditionalOptions) precondition (all_set (Dstat, Astat, TravelDate,
  TravelMoment))
    activity travel_advice
    ask ("Do you want to specify additional options?|")
    suggest('Yes', 'No')
    { YesNo (out AdditionalOptions)
      [ if (AdditionalOptions) // only select questions etc. beloning to activity 'additional_options'
          enable activity additional_options ;
        else // irrelevant => will not be asked:
          set_irv (FreeSubscription, InclReservationRequired, NeedExtraTransferTime, ExtraTransferMinutes)
      ]
    }

    question AskExtraTransferMinutesRequired (NeedExtraTransferTime) after AskAdditionalOptions
    activity additional_options
    ask "Do you need extra time for changing trains?|Extra time for changing trains?"
    suggest('Yes', 'No')
    {
      YesNo (out NeedExtraTransferTime)
      [if (!NeedExtraTransferTime) set_irv (ExtraTransferMinutes)] // irrelevant => will not be asked
    }


    question AskExtraTransferMinutes (ExtraTransferMinutes) after AskAdditionalOptions
    precondition (NeedExtraTransferTime)
    activity additional_options
    ask "How many extra minutes dou you need?"
    suggest(5, 10, 15, 20)
    {
      @ExtraTransferMinutes:<int>    // <int> matches any integer, <int:n..m> between n and m inclusive
      [ if (ExtraTransferMinutes > 20)
          fail("20 minutes extra is maximum|20 is maximum")
      ]
    }

    // The notation @varname:pattern means that variable ‘varname’ will be set to the value of ‘pattern’.
    // Example: @s:(john mary+ fred) will be set to “john mary mary fred” if that was what the user said.

    question AskFreeSubscription (FreeSubscription) after AskExtraTransferMinutesRequired
    activity additional_options
    ask "Do you have a Altijd Vrij of Trein Vrij subscription?|"
    suggest('Yes', 'No')
    {
      YesNo (out FreeSubscription)
    }


    question AskReservationRequired (InclReservationRequired) after AskFreeSubscription
    activity additional_options
    ask `Do you want trains for which reservation is required to be taken into account?|
        Take into account trains for which reservation is required?`
    suggest('Yes', 'No')
    {
      YesNo (out InclReservationRequired)
    }


    trigger SpecsCompleet1 () precondition (!any_unk(*)) // no more unknowns, enough questions answered
    activity travel_advice
    {
      DeterminePossibleJourneys (Dstat [1], Astat [1], null, travelDateTime, D_or_A, ExtraTransferMinutes,
        InclReservationRequired, FreeSubscription, true, out nrOfAlternatives, out curAlternative, out prices);

      if (nrOfAlternatives == 0)
      {
        say (`Unfortunately I cannot find a journey at the time you provided|No journey found`);
        say ("Please try again by specifying another time of travel");
        set_unk (D_or_A, TravelDate, TravelMoment)
        get_journey_plan = false;
      }
      else
      {
        optRmhNo = curAlternative; // optimal alternative

        if (nrOfAlternatives == 1)
          say ("I found @!<1> possible journey"); // @!<text>: text is emphasized
        else
          say ("I found a number of possible journeys");

        get_journey_plan = true;
      }
    }


    trigger ProvideJourneyDetails () precondition (get_journey_plan)
    activity travel_advice
    {
      string dTime;
      string aTime;
      string platformNo;

      ObtainJourneyDetails (curAlternative, out dTime, out aTime, out platformNo, out nrOfTrainChanges, out
  interchangeStations, out journeyTime, out journeySchedHtml, out journeySchedText, out interchangeHtml, out
  interchangeText);
      card ("Possible journey", null, journeySchedText, null, null); // BasicCard

      if (platformNo != null)
        say ("You will depart from {0} at {1} from platform {2}", Dstat [0], dTime, platformNo);
      else
        say ("You will depart from station {0} at {1}", Dstat [0], dTime);

      say ("You will arrive at station {0} at {1}", Astat [0], aTime);

      if (nrOfTrainChanges == 0)
        say "You do not have to change trains";
      else
        say ("You have to change trains {0} times", nrOfTrainChanges);

      if (journeyTime != null)
        say ("Your journey will take {0}", journeyTime);

      reported = true;
      get_journey_plan = false;
    }


    question AskRepeatJourneyDetails () precondition (reported)
    ask (nrOfTrainChanges == 0
         ? "Say @300 @!<earlier>, later, repeat or price"
         : "Zeg @300 @!<earlier>, later, repeat, transfers or price")
    suggest( nrOfTrainChanges == 0
             ? ['earlier', 'later', 'repeat', 'price', copyright]
             : ['earlier', 'later', 'repeat', 'transfers', 'price', copyright])
    { ( ( repeat | again )
        [get_journey_plan = true; reported = false ]
      | ( earlier
          [if (curAlternative == 0)
```

```
            fail ("This was the earliest of the {0} journeys found", nrOfAlternatives);
           say ("Now I will give an earlier journey");
           curAlternative --
          ]
      | later
         [if (curAlternative == nrOfAlternatives - 1)
            fail ("This was the latest of the {0} journeys found", nrOfAlternatives);
           say ("Now I will give a later journey");
           curAlternative ++
          ]
      )
      [ reported = false; get_journey_plan = true ]
    | ( stop | halt | done | thanks | thank you )
      [ user_done = true ]
    )
  }

  // Patterns -- basically: reusable regular expressions that can have input and output parameters

  pattern StationName (out string [] naam)
  {
    [ string [] s]
    station? @s:<any>{1,4}
    [ naam = CheckStation (s); if (naam == null) fail ($geenStation, s)]
  }


  // Date, Time etc. patterns: translated from Dutch more or less literally, not optimized for English

  pattern Date (out DateTime date, out string dateString, out string dayName)
  {
    [int dayNo = -1; int offset; int weeksFromNow = 0]
    ( ( today [offset = 0]
      | tomorrow [offset = 1]
      | the? day after tomorrow [offset = 2]
      )
      InNnnWeeks (out weeksFromNow)?
      [date = CalcDate (offset, 0, weeksFromNow, <year>)]
    | this?
      NameOfDay (out dayNo)
      [date = CalcDate (-1, dayNo, weeksFromNow, <year>)]
    | NameOfDay (out dayNo)
```

```
      next week // interpretation: first <...day> after this Sunday (Sunday can be today)
      [date = CalcDate (-3, dayNo, weeksFromNow, <year>)]
    | NameOfDay (out dayNo) // interpretation: first <...day> (can be today) plus <weeksFromNow> weeks
      InNnnWeeks (out weeksFromNow)?
      [date = CalcDate (-1, dayNo, weeksFromNow, <year>)]
    | on?
      MonthDate (out date)
    )
    [ dateString = DateToString (date);
      dayName = DayName (date)
    ]
  }


  pattern NameOfDay (out int dayNo)
  {
    ( sunday    [dayNo = 0]
    | monday    [dayNo = 1]
    | tuesday   [dayNo = 2]
    | wednesdag [dayNo = 3]
    | thursday  [dayNo = 4]
    | friday    [dayNo = 5]
    | saturday  [dayNo = 6]
    )
  }


  pattern InNnnWeeks (out int n)
  {
    ( in a week   [n = 1]
    | in @n:<int> weeks
    )
  }


  pattern MonthDate (out DateTime date)
  {
    [int yearNo = <year>, monthNo, dayNo]
    ( @dayNo:<int:1..31>
      ( januari  [monthNo = 1]
      | maart    [monthNo = 3]
      | mei      [monthNo = 5]
```

```
        | juli     [monthNo = 7]
        | augustus [monthNo = 8]
        | oktober  [monthNo = 10]
        | december [monthNo = 12]
        )
      | @dayNo:<int:1..29>
        februari [monthNo = 2]
      | @dayNo:<int:1..30>
        ( april     [monthNo = 4]
        | juni      [monthNo = 6]
        | september [monthNo = 9]
        | november  [monthNo = 11]
        )
      )
      ( @yearNo:<int:1500..> )?
      [? monthNo == 2 && dayNo == 29 && IsLeapYear(yearNo) || monthNo != 2 || dayNo != 29 ?]
      [date = CalcDate (-2, dayNo, monthNo, yearNo)]
    }


    pattern Moment (bool isToday, out DateTime time, out string timeString)
    {
      [ hhh = <hour>;
        mmm = <minute>;
        int n;
        int dayPart = -1 // -1 => not (yet) specified
      ]
      ( ( at? ClockTime (out hhh, out mmm)
          PartOfDay (out dayPart)?
        | PartOfDay (out dayPart)
          at? ClockTime (out hhh, out mmm)
        )                                     // suppose now 15:00 and hhh = 9:00 => hhh = 21:00
        [ if (isToday && dayPart == -1 && hhh <= 12 && <hour> > hhh ||
             hhh <= 6 && dayPart == 1 ||              // 2 o'clock in the afternoon => hhh = 14:00
             hhh >= 6 && hhh < 12 && dayPart == 2)    // 8 o'clock in the evening => hhh = 20:00
               hhh += 12;
        ]
      | [?isToday?] // guard, fails if date of travel is *not* today (‘now’ etc. pointless in that case)
        ( now
        | in
          MinutePeriod (out n)
          [hhh += n]
        //| MinutePeriod (out n)
        //  ago
        //  [hhh -= n]
        )
      )
      [ time = MakeDateTime (<today>, hhh, mmm);
        timeString = TimeToString (time);
      ]
    }


    pattern ClockTime (out int h, out int m)
    {
      [m = 0]
      ( ( quarter (to [m = -15] | past [m = 15] )
        | half [m = -30]
        | @m:<int:1..29> ( minute | minutes )?
          ( before [m = -m]
          | ( past | after )
          )
        )
        @h:<int:1..12>
        [ if (m < 0) {h -= 1; m = 60+m}]
      | @h:<int:0..24>  ( hours | ':' )
        ( @m:<int:0..59> ( minute | minutes )? )?
      )
    }


    pattern MinutePeriod (out int m)
    {
      [int n]
      ( ( @n:<int> | one [n = 1] )
        ( minute-s [m = n]   // minute-s will expand to ( minute | minutes ). Regexes are supported as well
        | ( hour | hours ) [m = 60*n]
        | quarter (of an hour)? [m=15*n]
        )
      | ( a half hour | half an hour ) [m = 30]
      | ( 1 | one ) and a half hour [m = 90]
      )
    }

```

```

pattern PartOfDay (out int dayPart)
{
  ( ( this | in the ) morning [dayPart = 0]
  | ( this | in the ) afternoon [dayPart = 1]
  | ( tonight | at night ) [dayPart = 2]
  )
}


pattern YesNo (out bool b)
{
  [string what]
  ( yes <any>*
    [b = true; say ("I heard yes")]
  | no <any>*
    [b = false; say ("I heard no")]
  | @what:<any>+
    [ b = false;
      fail ("I heard {0}. Please say yes or no|Say yes or no", what)
    ]
  )
}

// Commands -- when an answer to a question is received, the dialog driver
//             first checks whether the answer is a command. If it is not, an
//             applicable follow-up question will be searched

command Directives ()
activity travel_advice, additional_options
{
  ( ( i want )?
    ( a ( other | different ) | to? change )
    ( my | the )?
    ( departure station
      [?is_set (Dstat)?] // does only make sense if supplied earlier
      [set_unk (Dstat);
      ]
    | ( arrival | destination ) station
      [?is_set (Astat)?]
      [set_unk (Astat);
      ]
    | date | ( date of ( departure | arrival ) | ( departure | arrival ) date )
      [?is_set (TravelDate)?]
      [set_unk (TravelDate); set_unk (TravelMomentDA);
      ]
    | time
      [?is_set (TravelMoment)?]
      [set_unk (TravelMoment); set_unk (TravelMomentDA);
      ]
    | ( time of departure | departure time )
      [?all_set (TravelDate, TravelMoment)?] // does only make sense if supplied earlier
      [D_or_A = true; set_unk (TravelDate); set_unk (TravelMoment); set_unk (TravelMomentDA);
      ]
    | ( time of arrival | arrival time )
      [?all_set (TravelDate, TravelMoment)?]
      [D_or_A = false; set_unk (TravelDate); set_unk (TravelMoment); set_unk (TravelMomentDA);
      ]
    )
  | ( stop | done | halt )
    [ user_done = true ]
  | ( how long ( will | does ) the journey ( take | last ) // regexes are optimized, so don't
    | how long will i be on the way                        // worry about common start sequences
    )
    [ if (nrOfAlternatives == 0)
        say "No journey has been determined yet"
      else
        say ("Your journey will last {0}", journeyTime)
    ]
  | ( transfers
    | ( how ( often | many times )? do i have to change trains )
    )
    [ if (nrOfAlternatives == 0)
        say "No journey has been determined yet"
      else  if (nrOfTrainChanges == 0)
        say "You do not have to change trains";
      else
      {
        say ("You will have to change trains {0} times", nrOfTrainChanges);
        say ("You change trains at {0}", interchangeStations)
        card (Format ("Interchange stations ({0})", nrOfTrainChanges), null, interchangeText, null, null);
      }
    ]
  | [?prices != null && nrOfAlternatives > 0?] // guarded expression, must be true for this branch to succeed
```

```
        ( ( price | prices )
        | ( how much | what ) does ( it | ( the | this ) journey ) cost
        )
        [ if (FreeSubscription)
            say ("You have a subscription and there will be no additional costs|")
          else
          {
            say ("A 2nd class return ticket costs {0}| ", prices [1][0]);
            say ("A 2nd class one-way ticket costs", prices [1][1]);
            say ("A 1st class return ticket costs", prices [0][0]);
            say ("A 1st class one-way ticket costs {0}", prices [0][1]);
            table // show basic table with price info
            (
             "Prices for this journey",
             "prices are without reduction",
             [
               { header: 'Class', align: 'LEADING' },
               { header: 'Return', align: 'TRAILING' },
               { header: 'One-way',  align: 'TRAILING' }
             ],
             [
               { cells: ['2nd', prices [1][0], prices [1][1]] },
               { cells: ['1st', prices [0][0], prices [0][1]] }
             ],
             null,
             null
            )
          }
        ]
        )
      | help
        [ say("The following commands are available");
          say("Previous or back @1 To repeat the previous question"); // @1 is one second break
          say("Restart @1 To start all over");
          say("Stop @1 To terminate the session");

          if (nrOfAlternatives == 0)
            say("@400 When a journey has been determined, the following commands are available")

          say("Repeat @1 For the departure and arrival time");
          say("Transfers @1 For information about changing trains");
          say("Prices @1 For information about prices");
        ]
      )
    }

    command SystemCommands ()
    {
      ( start
        ( stop | halt )
        [say ("The session has been stopped"); halt]
      | ( (repeat the?)? previous question? | back )
        [redo]
      | restart
        [ say ("The session will restart");
          restart
        ]
      | ( speakeasy | maker | copyright | @copyright )
        [ say('This application has been developed by Speak Easy Software. @3|SpeakEasy Software')
          image ("https://www.speakeasy.nl/images/speakeasy_logo.jpg", "SpeakEasy Software Amersfoort");
        ]
      | version
        [say('The version number is <say-as interpret-as="ordinal">{0}</say-as>', <version>)]
      )
    }
}
```