`libtour` Manual

Viktor Pavlenko

Table of Contents

1 Overview

It is assumed that any sporting tournament can be described uniformly so that a generic software can be used to interpret the description.

Every sporting competition has participants (teams etc). Its rules define the number of stages of the competition and the groups of participants that play at each stage. Participants can advance to a next stage based on their performance during the previous stage(s). Game schedule defines when the games are played, and who plays them. Interpreting game results makes it possible to initialize subsequent stages and finally determine tournament winners.

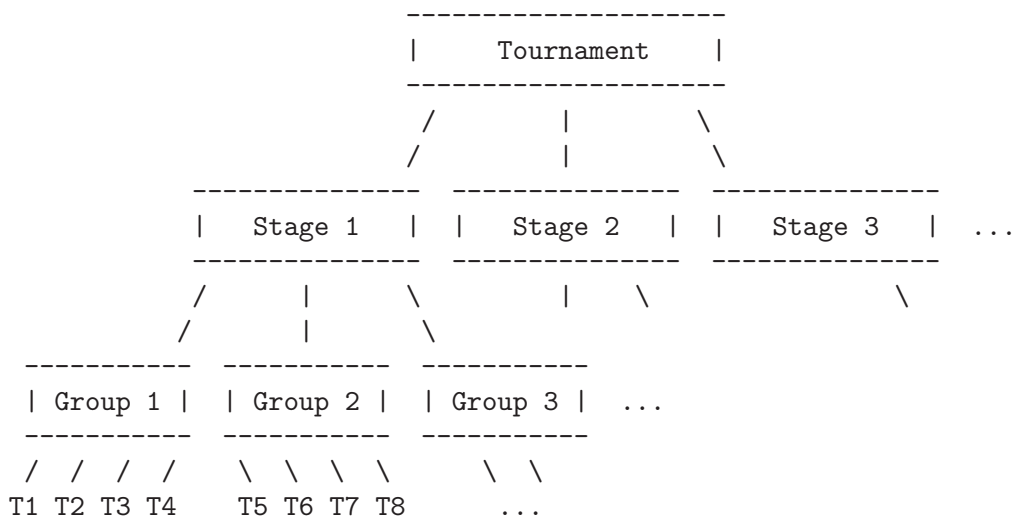
The following data constitute a tournament definition:

1. Rules
2. Teams
3. Schedule

`libtour`, a generic tournament processing engine, is written in C++ and compiled as a shared library. Tournaments are described in the Scheme programming language and are made available to the C++ engine by means of the Guile interpreter.

2 Structure of a tournament

Each tournament consists of one or more stages. Each stage has one or more groups of teams.



For example, 2002 FIFA World Cup tournament was played in five stages. The Preliminary stage had eight groups, from "A" to "H". Group "A" consisted of for teams (France, Senegal, Uruguay, and Denmark) etc. Text in parentheses corresponds to the elements of the above diagram.

- Preliminary (Stage 1)
  + A (Group 1)
    - France (T1)
    - Senegal (T2)
    - Uruguay (T3)
    - Denmark (T4)
  + ...

While the following stages and groups within were known in advance, the teams that would make it to them were not.

- Round of 16 (Stage 2)
  + Pair 1 of Round of 16
    - 1st place in group "A" of the Preliminary stage
    - 2nd place in group "F" of the Preliminary stage
  + ...

To be able to describe the groups of such an "semi-defined" stage we need to refer to its teams somehow. The solution is to make up some team IDs now, and establish the correspondance to the real teams later, when there is enough information to do so (e.g. when the previous stage completes). In our example we can assign an ID of "S1-A-1" to

the team at “1st place in group “A” of the Preliminary stage”, “S1-F-2” to one at “2nd place in group “F” of the Preliminary stage” and so on.

Game schedule for a stage may not be known either, until a previous stage completes. In `libtour` it is assumed that

- Real team ID, and
- (a part of) Game schedule

are the only missing pieces of information for a following stage, and they can be determined by the tournament rules on the completion of the previous stage(s).

All other information needed for a tournament definition does not require a delayed computation and can be statically provided by the rules. The list below shows the most important items.

- Fields for table of standings (“Games Played”, “Games Won”, “Goals Scored” etc.)
- Fields for a game results (“Goals Scored”, “Goals Allowed” etc.)
- How to interpret a game result
- How to sort teams in a group

A team can be a member of more than one group. For instance, in an NHL competition each team belongs to a conference and a division group at the same time. Moreover, when a stage consists of several groups, it is often convenient to view the combined results of all teams; such a summary group can be automatically created by `libtour`.

3 Installation

IMPORTANT: Make sure to `make uninstall` a previous `libtour` installation and remove '`$HOME/.libtour`' directory if any.

You need a `g++` compiler and `guile 1.6` to compile and use this software. Please refer to the '`INSTALL`' file for the complete installation procedure.

`libtour` headers will be installed in '`$prefix/include`', the libs in '`$prefix/lib`' and the Scheme tournament definition files in '`$prefix/share/libtour/scheme`'. To create an HTML documentation (in addition to the info files) supply an `--enable-htmldoc` switch to the `configure` script.

4 Using the CLI client

Complete Scheme definitions are provided for the following tournaments:

- 2002 FIFA World Cup, module `(tour-defs fifawc2002)`
- 2001/2002 NHL, module `(tour-defs nhl2002)`
- 2002/2003 NHL, module `(tour-defs nhl2003)`
- 2003/2004 NHL, module `(tour-defs nhl2004)`
- 2003 Ice Hockey WC, module `(tour-defs ihwc2003)`

`tourcli` program is limited to loading a single tournament at time.

4.1 Quick start

To start the client for the 2002 FIFA World Cup tournament, run the client with module name as the sole argument:

```
$ tourcli fifawc2002
```

and it will load the fifawc2002 configuration from '`$prefix/share/libtour/tour-defs/fifawc2002.scm`'. To search a different directory before the standard one for modules use `-d` command line switch:

```
$ tourcli -d dir-with-my-modules fifawc2002
```

Once in the command loop, type `h q` or `h i` for list of available commands (read-only and modifying queries correspondingly). To load the results of the tournament from the batch file provided type `l $prefix/share/libtour/data/fifawc2002_res.txt`. If you added or removed game results, you can choose to save your changes with 's' command or at exit. Updated results will be saved in '$HOME/.libtour/fifawc2002_results.scm'. Next time `tourcli` is started for the 2002 FIFA WC, it will restore its state.

For examples of usage of the CLI application See Appendix A [CLI client sample session], page 47.

4.2 Usage reference

Help:

h [q|i] print usage

Queries, q command:

q i print tour info

q s [-s <stage-id>]
print stage(s)

q g -s <stage-id> [-g <group-id>]
print group(s) for stage <stage-id>

q t -s <stage-id> [-g <group-id>]
print teams for stage <stage-id> group <group-id>

q k -s <stage-id> [-g <group-id>] [-m <game-id-list>]
print schedule for stage <stage-id> [group <group-id>], showing only game IDs <game-id-list> if provided

q p -s <stage-id> -g <group-id>
print standings for stage <stage-id> group <group-id>

q sc -s <stage-id> [-g <group-id>]
print sorting conflicts for stage <stage-id> group <group-id>

q scd -s <stage-id> -g <group-id> -pos <position-range>
print details for a sorting conflict for team positions <position-range> in stage <stage-id> group <group-id>

q x -s <stage-id> [-g <group-id>]
print excessive game ids for stage <stage-id> [group <group-id>]

q b -s <stage-id> -m <game-id>
print groups that contain game <game-id> for stage <stage-id>

q tmf print team fields

q stf -s <stage-id>
print standings fields for stage <stage-id>

q grf -s <stage-id>
print game result fields for stage <stage-id>

q scf print schedule fields

q df print date format used in schedule

q hp print position of the home team

q w print winners

q ft -s <stage-id>
print available team filters for custom groups

q fg -s <stage-id>
print available game filters for custom groups

q fa -s <stage-id> -(ft|fg) <filter name>
print required arguments for a team or game filter

mp -s <stage-id>
print maximum point in game for stage <stage-id>

Inserts, i command:

i m -m <game-id> -r "<game-res>" [-s <stage-id>]
insert or replace result <game-res> for game <game-id>

i d -m <game-id> [-s <stage-id>]
delete result of game <game-id>

i c -s <stage-id>
set stage <stage-id> to be complete

i uc -s <stage-id>
: set stage <stage-id> to be incomplete (invalidates previous stages!)

i g -s <stage-id> -g <group-id> -id <new-group-id> [-n <new-group-name>] [-ft <team-filter>] [-fg <game-filter>]
: create a group with ID <new-group-id> within stage <stage-id> after group <group-id> using <team-filter> and/or <game-filter>

i dg -s <stage-id> -g <group-id>
: delete a group with ID <group-id> within stage <stage-id>

i sc -s <stage-id> -g <group-id> -pos <position-range> -sol <team list>
: resolve sorting conflict between teams in <position-range> in group <group-id> of stage <stage-id>

Loading/saving results:

l filename
: load results from the batch file '`filename`'

s
: save results of current tournament in '`$HOME/.libtour/`' directory

5 The engine

A number of C++ classes mimic the components of a sporting tournament described in See Chapter 2 [Structure of a tournament], page 3.

This chapter describes main classes of the `libtour` implementation and their collaboration. It is meant to provide an implementation overview but also should serve as a high level explanation of how `libtour` works. For details please refer to the C++ source in '`libtour-N.N/libtour/`' directory.

5.1 Main classes

`TTour` class represents a tournament. It contains:

- a collection of stages (`TStage` objects)
- a collection of teams (a map of `TTeam` objects where keys are global team IDs)

`TTour` provides an interface for looking up teams and stages as well as encoding its state in a Scheme data structure, and restoring itself from the saved state (see Section 5.8 [Saving and restoring state], page 17).

`TStage` class is a representation of a tournament stage. It contains:

- a collection of groups (`TGroup` objects)
- a collection of teams (a map where keys are teams IDs local to this stage, and values — pointers to the `TTeam` objects stored in the `TTour` instance)
- an array of games (`TGame` objects) with optionally set results

Some other stage parameters are configurable from the Scheme definition:

- a thunk for each team, when run will return a real team ID
- thunk(s) for game schedule, when run will return a list of game specifications
- a procedure to interpret game results
- format specifications for team result in standings, and game result

`TStage` objects keep record of their state. See Section 5.3 [TStage state transitions], page 13, for detailed description of `TStage` state transitions.

`TGroup` class corresponds to a group of teams in a tournament that usually play games between one another and have their results summarized in a table of standings. A group (`TGroup`) only exists within a stage (`TStage`).

`TGroup` knows its teams and keeps track of their performance. The Scheme tournament definitions must provide each group with the following information:

- how to compare two team results (used for sorting)
- how to determine if a game belongs to this group
- how to determine if all games for this group have been played (if the group is "complete")

`TFormula` class hierarchy gives the groups capability to determine if all their games have been played (if the groups are `"complete"`). See Section 5.4 [Game results and formula], page 14, for more.

TGroup object also stores information about sorting conflicts if any (see Section 5.7 [Sorting teams], page 17).

TTeam objects represent participants of a competition. They are created and stored in the TTour instance. Teams are known by their global ID across the tournament, and by stage-local ID to the stages and groups.

Each TGroup object contains TTeamRes instances for each team that belongs to the group. TTeamRes accumulates interpreted game results for a team and can be thought of as a representation of a row in the table of standings.

TGame class represents a game between two teams in a tournament. Each of them contains a pointer to TGameRes which is zero until the game result is known. The game is read-only (i.e. its result cannot be changed) if its result was specified in the Scheme schedule definition (see Section 6.5 [Game schedule definition], page 26).

TQuery class provides the means for the user code to interact with the libtour engine. It only keeps a reference to the TTour instance and uses its public interface to get hold of TStage, TGroup, and other objects as required.

Through a TQuery instance it is possible to query or change the tournament state. In each case, a T_QuerySpec object must be built and passed to TQuery for processing. For a read-only request (a query), a TQueryResults instance is filled with the reply. The user is provided with T_xxx envelope classes for TQuery and TQueryResults that are described in See Chapter 8 [libtour API], page 33.

When an error occurs an instance of T_Exception class hierarchy is thrown. In case of libtour internal problem, the exception is T_InternalException. Invalid requests, or errors in tournament definitions (Scheme code) will lead to T_RuntimeException. T_StageReadyException (a descendant of T_RuntimeException) will be thrown if a stage completion request could not be performed because of the following stage not being able to become ready (see Section 5.3 [TStage state transitions], page 13).

5.2 Scheme/C++ interaction

During a tournament interpretation libtour will read data structures defined in Scheme and call procedures embedded in them. These Scheme procedures can use C++ static functions exported back into Scheme.

TUsrProcedures singleton provides C++ functions for the Guile interpreter. To be able to locate the proper tournament when receiving a call from Scheme, TUsrProcedures requires each TTour object to register before the exported functions can be used. Naturally, each exported C++ function requires a tournament name as its argument so that the corresponding TTour object can be located.

The C++ functions available from Scheme are exported into the (libtour builtin) Guile module. See Section 7.1 [builtin module], page 27, for detailed description of '(libtour builtin)'.

C++-defined procedures are guaranteed to throw proper Scheme errors when a C++ exception is thrown by the C++ code. This "C++ exception -> Guile error" conversion is implemented as shown below:

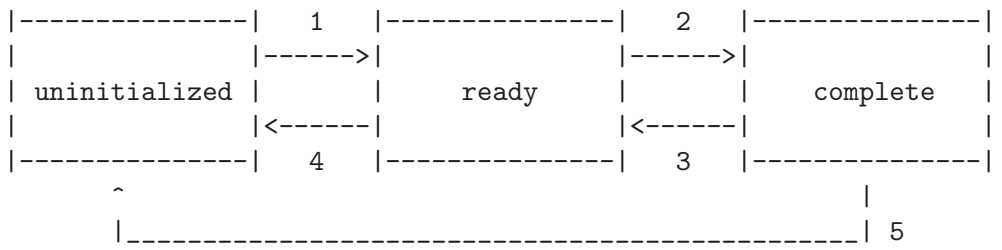
```
{
    SCM ret_scm = scm_listify( SCM_UNDEFINED );
    try {
        //do processing, set ret_scm

    } catch ( T_Exception& e ) {
        scm_throw( scm_str2symbol("t_exception"),
                   scm_makfrom0str(("<C++ function name>: " +
                                    e.why()).c_str()) );
    }
    return ret_scm;
}
```

Likewise, Guile errors, possibly caused by syntatically invalid scheme code or thrown by the Scheme procedures called from C++, are handled with `scm_internal_catch` and converted into C++ exceptions. See `TCallWithCCExcp` class implementation for details.

5.3 TStage state transitions

`TStage` class delays determining it's teams (and complete game schedule) until the previous stage is complete. There are three possible states of a `TStage` object:

- uninitialized — no teams (and possibly games) are known, the team map (see Section 5.1 [Main classes], page 11) contains zeroes as values, and the games collection contains only the games statically defined in the schedule
- ready — team map values point to the real `TTeam` objects, and game schedule usually contains games
- complete — all games have been played



On the "uninitialized -> ready" transition (1), teams (and sometimes game schedule) are determined by calling the user-supplied Scheme procedures (thunks). `TStage` that is trying to become "ready" (1) runs an SCM thunk for each local team ID to locate the corresponding `TTeam` object, and asks its `TSchedule` member for a list of the games (this will run more Scheme code if schedule is not static). If the Scheme procedure fails to return an unambiguous result (because the previous stage is not complete, or a team sorting conflict prevents exact team positioning, etc) a `T_Exception` will be thrown.

To become "complete" (transition 2) a stage relies on its groups (`TGtoup`) when deciding if all games have been played; in other words, a stage is complete if all its groups are complete. Completeness of a group is decided according to its playing "formula" (see Section 5.4 [Game

results and formula], page 14). Another condition must be satisfied for a stage to become "complete": if this stage is not the last in the tournament, the following stage should be able to become "ready" (transition 1). If this fails (usually due to a sorting conflict in the previous stage), `libtour` will throw a `T_StageReadyException` (a descendant of `T_RuntimeException`.

A stage in the "complete" state cannot be changed, except for adding and deleting filtered groups (see Section 5.6 [Filtered groups], page 15).

It is possible for a stage to undergo opposite transitions: a "complete" stage can become "ready" by executing `uncompete-stage` modifying query (see Section 8.5.2 [Modifying queries], page 44).

Whenever a stage ceases to be "complete", all following stages in "complete" or "ready" state automatically become "uninitialized" (transitions 4 and 5).

Note that during loading saved state (see Section 5.8 [Saving and restoring state], page 17) stages become "complete" automatically. In all other cases issuing the `compete-stage` modifying query (see Section 8.5.2 [Modifying queries], page 44) is necessary.

5.4 Game results and formula

A `TQuery` object passes a game result to the corresponding stage for processing.

The stage will find the `TGame` object, create its result (`TTGameRes`) and ensure that the game belongs to at least one group and no group reject it for some reason (decided by the groups' `TFormula` members). If this check passes, the game result will be communicated to the groups and set in the `TGame` object. This two-pass process is necessary to keep the groups in consistent state.

Game result updates and deletions are processed similarly.

There are a number of predefined types of the group playing formulas implemented as `TFormula` class hierarchy:

- "round robin"
- "preset number of games for each team"
- "best-of N series"
- "none"

"none" formula has no way to determine the group's completeness independently and relies on the groups within the stage which have a real (non-"none") formula.

In the "best-of N series" (playoff) formula the schedule may contain excessive games; indeed, if the teams play until four wins, the last three of the seven scheduled games may not be necessary. A game result will be rejected for an excessive game. Some restrictions apply on game result update and deletion.

`TGroup` objects can also report a list of IDs of their excessive games which also works for groups with "none" formula.

5.5 Summary group

If the `SUMMARY` entry of a stage definition (see Section 6.3.5 [Summary definition], page 25) is a procedure, `libtour` will create a summary group for that stage. The summary group will be defined as follows:

ID "ALL"

NAME "All teams"

G-TEAMS all teams that belong to the stage

FORMULA `NONE`

G-CMP-F the procedure given in `SUMMARY` stage definition field (see Section 6.3.4.3 [Teams comparison procedure], page 23)

G-GAME-HERE-F
`ANY`

5.6 Filtered groups

Groups of teams within a stage are defined in the tournament's rules (see Section 6.3.4 [Groups definition], page 22). It is also possible to create a group on the fly using

- team filters, and/or
- game filters

The idea behind filtered groups is as follows. A filtered group is based on a pre-defined group (one in the rules definition). By using filters the user can filter out teams and/or games that belong to the base group at the time of creation of the filtered group.

For each filter a filter-making procedures should be defined. These procedures are written in Scheme. A number of them are provided by `(libtour filters)` module, but also can be defined directly inside tournament's rules.

A team/game filter-making procedure takes at least three arguments:

- tournament name
- stage id
- new group id

Following arguments if any are used to configure the filter. They should be provided by the user. `libtour` will supply the first three arguments when calling the filter-making procedure.

A team filter-making procedure will retun a Scheme closure (so that all the parameters are remembered) that takes a team associative list as its only argument. The list has keys as in `STANDINGS-FIELDS` (see Section 6.3.1 [Standings and Game result fields], page 21).

A game filter-making procedure returns a closure which is like a game-belongs-to-group procedure in a group definition (see Section 6.3.4.4 [game-belongs-to-group procedure], page 24). It expects a single argument of a game associative list with keys as in `SCHEDULE-FORMAT` (see Section 6.5 [Game schedule definition], page 26) and returns a pair of boolean values, one for each team.

Documentation of every filter-making procedure as well as the resulting closure must be provided as they are used by libtour

'filter-making procedure documentation'
: documentation string (the first string expression in the definition)

'filter-making procedure's 'args property'
: associative list of argument names and their types, for instance

```
'(("field" . symbol)
  ("action" . symbol)
  ("values" . list))
```

The supported argument types are defined in 't_constants.hh' header and include

- bool
- int
- string
- symbol
- list

'filter procedure's 'docs property'
: string desribing what is being filterd out

If a team filter is provided, libtour will call it for every team of the base group to determine which should also belong to the filtered group. Once the group is created, libtour will loop through the games of the base group and call the game filter (if any) to determine which of the games should also belong to the newly created group.

Based on the filter information, libtour will call the TGroup constructor with the following associative list of the group definition (see Section 6.3.4 [Groups definition], page 22):

ID provided by user

NAME provided by user, or "<base group id> (U)"

DESCRIPTION
: docs property of the filter procedure(s)

G-TEAMS base group's team IDs after applying the team filter if any

FORMULA NONE

G-CMP-F the procedure given in SUMMARY stage definition field (see Section 6.3.5 [Summary definition], page 25) if supplied, or the base group's comparison procedure (see Section 6.3.4.3 [Teams comparison procedure], page 23)

G-GAME-HERE-F
: game filter if supplied, or the base group's procedure (see Section 6.3.4.4 [game-belongs-to-group procedure], page 24)

It is important to note that a filtered group formula is always NONE (see Section 6.3.4.2 [Group formula definition], page 23).

When a stage containing filtered groups becomes uninitialized (see Section 5.3 [TStage state transitions], page 13), all filtered groups are deleted. When saving results (see Section 5.8 [Saving and restoring state], page 17) no sorting conflicts are recorded for filtered groups.

5.7 Sorting teams

The Scheme tournament definition provides a procedure for sorting teams in each group. Proper sorting of teams within a group is essential for (1) constructing of a table of standings, and (2) determining the teams that advance to the following stage and the winners of the tournament.

It is possible that exact positions of two teams can't be established (e.g. they both have equal scores according to the sorting procedure). In such case `TGroup` will record a "sorting conflict".

Sorting conflict are represented by the `TSortConflict` class and managed by the `TSConflicts` class. Each `TGroup` uses a `TSConflicts` instance to handle sorting conflicts.

Position of a team within a group is represented by a pair of unsigned integers. They are equal for a team without a sorting conflict. If, on the other hand, a team has a sorting conflict, the integers will differ. In the case it is essential to know the team at certain position (like in this scenario: "the team at first place in group G advances to the next stage"), the sorting conflict that prevents exact positioning will have to be resolved.

Resolution of sorting conflicts falls outside of the tournament rules defined in Scheme. `libtour` provides an interface for resolving sorting conflicts (see Section 8.5.2 [Modifying queries], page 44).

For comparing personal results between two teams (as it is often required by the tournament's rules), `TGroup` can create and delete a helper `TGroup` within itself. See Section 7.2 [common module], page 28, `*make-tmp-group*` for an example how temporary groups can be utilized.

5.8 Saving and restoring state

A user application can ask the `TTour` instance for a Scheme data structure that encapsulates its current state. This data structure can be used later to restore the state of the tournament.

The following information is stored for each stage:

- game results
- sorting conflicts

Here is the structure of the state list:

```
(<stage-list> ...)

stage-list:
("stage-id" <game-result-list> <sort-conflict-list>)

game-result-list:
(("game-id" . result-list) ...)

result-list: list of numbers as long as the game field list
(see Section 6.3.1 [Standings and Game result fields], page 21)

sort-conflict-list:
```

```
(<group-sc> ...)

group-sc:
("group-id" <sc> ...)

sc:
(("stage-local-team1-id" . <resolution>)
 ("stage-local-team2-id" . <resolution>) ...)

resolution:
-1 if the conflict is not resolved, 0 or greater to
notify relative position of the team within the conflict
```

When feeding the state data structure at the tournament loading time, `libtour` will pass the game results (see Section 5.4 [Game results and formula], page 14) as well as sorting conflict information to the stages so that the correct positioning of the teams within the underlying groups will be restored.

6 Scheme definitions

In order to interpret a sporting tournament under the `libtour` engine, one has to write a tournament definition in the predefined format in the Scheme programming language.

The following three pieces of definition must be provided:

- tournament rules (static contents of stages and groups, as well as Scheme procedures)
- teams that start the tournament, and
- game schedule

6.1 Defining a tournament module

To make a tournament definition suitable for the `libtour` engine, one has to write a file called '`<tour-name>.scm`' and put it in a directory where `libtour` will find it (for the '`tourcli`' CLI client, see Section 4.1 [Quick start], page 7). The file has to define a Guile module, `tour-defs <tour-name>` and export the following variables:

`<tour-name>:INFO`
: associative list for general information, `TITLE` key is required

`<tour-name>:TEAMS`
: team definitions (see Section 6.2 [Teams definition], page 20)

`<tour-name>:TEAMS-FORMAT`
: format of team definitions (see Section 6.2 [Teams definition], page 20)

`<tour-name>:STAGES`
: stage definitions (see Section 6.3 [Stages definition], page 20)

`<tour-name>:WINNERS`
: tournament winner definitions (see Section 6.4 [Winners definition], page 25)

`<tour-name>:SCHEDULE`
: schedule definitions (see Section 6.5 [Game schedule definition], page 26)

`<tour-name>:SCHEDULE-FORMAT`
: format of schedule definitions (see Section 6.5 [Game schedule definition], page 26)

`<tour-name>:SCHEDULE-DATE`
: format of date/time field in schedule definition (see Section 6.5 [Game schedule definition], page 26)

`<tour-name>:SCHEDULE-HOME`
: position of the home team in the schedule (see Section 6.5 [Game schedule definition], page 26)

The definitions are usually split into several files. For instance, here is a partial content of '`fifawc2002.scm`':

```
(define-module (fifawc2002))

(use-modules ((fifawc2002_rules))
```

```
                ((fifawc2002_teams)
                 :renamer (symbol-prefix-proc 'fifawc2002:))
                ((fifawc2002_schedule)
                 :renamer (symbol-prefix-proc 'fifawc2002:)))

    ;; definitions of INFO, STAGES, WINNERS omitted

    (export fifawc2002:INFO fifawc2002:STAGES fifawc2002:WINNERS)
    (re-export fifawc2002:TEAMS fifawc2002:TEAM-FORMAT
               fifawc2002:SCHEDULE fifawc2002:SCHEDULE-FORMAT
               fifawc2002:SCHEDULE-DATE fifawc2002:SCHEDULE-HOME)
```

The following sections describe contents of the exported variables.

6.2 Teams definition

An associative list pointed to by the `<tour-name>:TEAMS` variable binds global team IDs to the team descriptions.

```
<tour-name>:TEAMS :
'(("<TEAM-GLOBAL-ID-1" <team-description-list>) ...)
```

`<team-description-list>` is a list of strings as defined by the `<tour-name>:TEAM-FORMAT` variable. Only `NAME` of teams is required to be specified in `<tour-name>:TEAM-FORMAT`, this name will be returned as the team identifier in `libtour` queries.

For instance, team definitions for the 2003 Ice Hockey WC included in the `libtour` distribution, contains the following (trimmed for space):

```
(define ihwc2003:TEAMS
  '(("SVK" "Slovakia")      ; "SVK" is a global team ID
    ("RUS" "Russia")        ; "Russia" is a team name
                            ;   according to the TEAM-FORMAT
                            ;   definition below
    ("SWE" "Sweden")
    ...))

(define ihwc2003:TEAM-FORMAT
  '(NAME))
```

6.3 Stages definition

`<tour-name>:STAGES` variable exported by the tournament module contains a Scheme list where each list member describes one stage.

```
<tour-name>:STAGES :
(<stage-definition> ...)

<stage-definition>:
((ID . <string for the stage ID>)
```

```
(NAME . <string for the stage name>)
(STANDINGS-FIELDS . <standings-fields list>)
(GAMERES-FIELDS . <gameres-fields list>)
(RES-F . <game-result-interpretation-procedure>)
(TEAMS . <teams-definition>)
(GROUPS . <groups-definition>)
(SUMMARY . <summary-definition>)
(FILTERS . <filters-definition>))
(MAX-POINTS . <max-points-definition>))
```

The `ID` string can be anything, by convention it is "S1" for the first stage, "S2" for the second etc. The `NAME` string usually denotes what the stage is called, "Preliminary", or "Quarterfinals".

`MAX-POINTS` defines maximum number of points a team can get in a game. If such information can be provided for this stage, the value must be an integer greater than zero. If, on the other hand, maximum points can't be defined, a false value `#f` shoud be used.

The rest of the keys are described in the following subsections.

6.3.1 Standings and Game result fields definitions

`STANDINGS-FIELDS` defines a list of column (or field) names for a team in a table of standings, like this:

```
'(GAMES WINS DRAWS LOSSES GOALS-FOR GOALS-AGAINST POINTS)
```

All the fields will contain integers. The following fields are required to be present: `GAMES`, and `POINTS`

`GAMERES-FIELDS` denotes the fields for a game result. It consists of a pair of lists, the names of the fields in the first and types of the fields in the second. For instance, if a game results can be described by three items, goals scored by the first team, goals scored by the second team, and whether the game ended in overtime, the game fields may be defined like following:

```
'((GF GA OT) int int bool)
```

`int` and `bool` are the only two supported types.

6.3.2 Game results interpretation procedure

A Scheme procedure placed under the `RES-F` key will be used by `libtour` for interpreting a game result. It expects an associative list as the sole argument, whose keys are the same as `GAMERES-FIELDS`. For instance, for the `GAMERES-FIELDS` shown above and the game in which the first team lost 2 to 3 in regulation time (OT is false), the argument will be

```
'((GF . 2) (GA . 3) (OT . 0))
```

The `RES-F` procedure should return a pair of associative lists that describe change of `STANDINGS-FIELDS` for each of the teams. Continuing our example, if a team is awarded two points for a win, and zero for a loss, the procedure can return the following:

```
'(                          ;;; first team list
  ((GAMES . 1)               ; increment game count by 1
   (LOSSES . 1)              ; increment loss count by 1
   (GOALS-FOR . 2)           ; first team scored 2 goals
   (GOALS-AGAINST . 3))      ; first team allowed 3 goals

  (GAMES . 1)               ;;; second team list
  (WINS . 1)                 ; increment win count by 1
  (GOALS-FOR . 3)
  (GOALS-AGAINST . 2)
  (POINTS . 2))              ; second team earned 2 points
```

The return lists may contain an association pair for any of the fields found in the `STANDINGS-FIELDS` list even if their values haven't changed (such as `(POINTS . 0)` for the first team).

Note that values of the `POINTS` entry cannot exceed the value defined in `MAX-POINTS` for current stage (see Section 6.3 [Stages definition], page 20).

`mk-res-lists` helper procedure defined in the `(libtour common)` module (see Section 7.2 [common module], page 28) is useful when creating a resulting list.

6.3.3 Stage teams definition

The `TEAMS` definition is an associative list with team IDs local to stage (see Chapter 2 [Structure of a tournament], page 3) as keys and procedures of no argument (thunks) that return global team IDs when called:

```
'(("TEAM-LOCAL-ID-1" . <thunk>) ...)
```

Team local IDs can be any strings, but it is convenient to give them somewhat meaningful names. For the first stage of a tournament teams are known in advance, therefore the `TEAMS` list might look like following:

```
`(("TEAM-LOCAL-ID-1" . ,(lambda () "TEAM-GLOBAL-ID-1"))
  ...)
```

For the following stages the thunk will be called after the previous stage is compete. Here is an example of specifying the team that took the first place in group "A" in the previous stage:

```
`("PREV-STAGE-A-1" .
  ,(lambda () <code to determine the real team ID>))
```

The thunk has to throw a Guile error if it cannot determine the global team ID.

6.3.4 Groups definition

The `GROUPS` key in the stage definition list corresponds to an associative list of group definitions.

```
<group-definitions>:
(<group-definition> ...)
```

```
<group-definition>:
((ID . <string for the group ID>)
 (NAME . <string for the group name>)
 (DESCRIPTION . <string for the group description>) ;; optional
 (G-TEAMS . <stage-local team-IDs list>)
 (FORMULA . <playing formula specification>)
 (G-CMP-F . <teams-comparison-procedure>)
 (G-GAME-HERE-F . <game-belongs-to-this-group-procedure-or-spec>))
```

Like for a stage, the `ID` string can be anything unique, and `NAME` string usually describes what a group is, like "Group A", or "Semifinal Pair 1". The `DESCRIPTION` string is optional and can contain some additional information about the group; it is filled in by `libtour` for filtered groups (see Section 5.6 [Filtered groups], page 15).

6.3.4.1 Group teams definition

The `G-TEAMS` is a list of team IDs (local to the stage). Team IDs for a group should be subset of the the keys in stage's `TEAMS` definition (see Section 6.3.3 [Stage teams definition], page 22). A team may belong to more than one group within a stage.

6.3.4.2 Group formula definition

Group playing formula helps determine how many games should be played between the teams within a group and when the group becomes complete (see Section 5.4 [Game results and formula], page 14). The following values are allowed for a formula definition:

`'(ROUND <number>)`
: round robin with <number> of games to be played between each two teams

`'(TEAM-GAMES <number>)`
: each team will play <number> of games (similar to round robin but the number of games between any two teams is not enforced)

`'(PLAYOFF <number>)`
: for a group of two teams, games continue until one team has <number> of wins

`'(NONE)`
: no formula, the group will accept any number of game results as long as they are accepted by at least one group with a non-"none" formula within the stage

6.3.4.3 Teams comparison procedure

The procedure defined under the `G-CMP-F` key in the group definition will be used by `libtour` for sorting teams within the group.

The procedure must accept five arguments: (1) tournament name, (2) stage ID, (3) group ID, and (4,5) team result lists of the two teams to be compared. The first three arguments are obvious. Each of the last two contains an associative list of team results (see Section 6.3.1 [Standings and Game result fields], page 21) plus a team ID (local to stage) association pair, for example:

```
'((GAMES . 10)                ;; these are STANDINGS-FIELDS
  (WINS . 7)
  (DRAWS . 1)
  (LOSSES . 2)
  (GOALS-FOR . 22)
  (GOALS-AGAINST . 7)
  (POINTS . 15)
  (ID . "<TEAM-LOCAL-ID>"))   ;; additional ID entry
```

The procedure must return `1` if the first team results are higher than the second's, `-1` if opposite, and `0` if the teams appear to be equal.

An example of the sorting procedure below uses helper procedures from module `(libtour common)` (see Section 7.2 [common module], page 28).

```
(define cmp-f
  (lambda (tour-name stage-id group-id t1 t2)
    (or
        ;; compare points
        (cmp-assoc-field-p 'POINTS t1 t2)

        ;; if points are equal, compare goal difference
        (cmp-assoc-field-diff-p 'GOALS-FOR 'GOALS-AGAINST t1 t2)

        ;; if goal differences are equal, see which team scored more
        (cmp-assoc-field-p 'GOALS-FOR t1 t2)

        ;; if equal, compare personal games for these two teams
        (cmp-two-teams-p c-tour-name
                         stage-id
                         (assoc-ref t1 'ID)
                         (assoc-ref t2 'ID)
                         ;; another cmp procedure
                         cmp-prelim-stage-personal-f
                          ;; consider only games between these two teams
                         'BOTH)
        0)))  ;; give up, the teams are equal
```

6.3.4.4 game-belongs-to-group procedure

To determine if a game belongs to a given group, another Scheme procedure should be supplied. For the two most common cases, however, `libtour` provides shortcuts so that a predefined Scheme symbol can be given instead of a procedure.

`'BOTH` game belongs to the group if *both* teams that played belong to the group

`'ANY` game belongs to the group if *any* team that played belong to the group

`(lambda (game-alist) ...)`
a procedure which returns a pair of boolean values for each team; it is `(#f . #f)` if none of the two teams' result should be recorded in this group (and therefore

the game does not belong to this group), or one of `(#t . #f)`, `(#f . #t)`, `(#t . #t)` if only the first, or only the second, or both teams' results correspondingly should be recorded in this group.

If a procedure is supplied, it should expect a game associative list and be in the format defined by the `SCHEDULE-FORMAT` (see Section 6.5 [Game schedule definition], page 26), for instance:

```
'((ID . 123)
  (DATE . "23.06.2003 15:00")
  (TEAM1 . "<TEAM-LOCAL-ID-1>")
  (TEAM2 . "<TEAM-LOCAL-ID-2>"))
```

As mentioned, the procedure should return a pair of boolean values.

6.3.5 Summary definition

A `SUMMARY` entry controls creation of a summary group for a stage (see Section 5.5 [Summary group], page 15). If its value is `#f`, no summary group will be created. If it contains a procedure, a summary group will be created for the stage, and this procedure will be used for team comparison (like see Section 6.3.4.3 [Teams comparison procedure], page 23) within this group.

6.3.6 Filters definition

A `FILTERS` entry in a stage definition defines what team and/or game filters are aplicable to this stage (see Section 5.6 [Filtered groups], page 15). It contains an associative list with the following optional keys: `TEAM-FILTERS`, and `GAME-FILTERS`; the values are lists of filters:

```
((TEAM-FILTERS <team-filter-making-procedure> ...)
 (GAME-FILTERS <game-filter-making-procedure> ...))
```

If no filters can be applied, the list is empty.

6.4 Winners definition

Winners (or losers) are the teams that finished the tournament with some distinguishable result: won gold, or lost everything. Winners definition is very similar to the one for teams within a stage (see Section 6.3.3 [Stage teams definition], page 22). The differences are:

1. a string describing what the team won is used in the place of stage-local team ID
2. if the team cannot be determined the thunk should return `#f` rather than throw error.

Here is an example of a Winners list:

```
(define <tour-name>:WINNERS
  '(("Gold" . <thunk>)
    ("Silver" . <thunk>)
    ("Bronze" . <thunk>)))
```

6.5 Game schedule definition

The <tour-name>:SCHEDULE variable contains a list in the form:

```
'(<stage-list> ...)

stage-list:
("stage-id" <entry> ...)

entry:
<game-definition> or <thunk>
```

where <game-definition> is a list of string members defined in tour-name:SCHEDULE-FORMAT, and <thunk> is a procedure of no arguments that returns a list of <game-definition>. The <thunk> list elements (if any) are run at the time the stage changes its state from "uninitialized" to "ready".

A <game-definition> may look like this:

```
'(("1" "31.05.02 20:30" "<place>"
   "<stage-local team ID 1>" "<stage-local team ID 2>"))
```

To allow loading stages with unknown schedule <entry> members are optional. Note, however, that a stage with no schedule can never become "complete".

tour-name:SCHEDULE-FORMAT variable defines the meaning of the list members. For the preceding example it would be:

```
'(ID DATE TIME PLACE TEAM1 TEAM2)
```

Note that ID, DATE, TEAM1, and TEAM2 fields are mandatory, any other — optional. Games are not required to be sorted as the sorting is done internally by libtour.

Interestingly, you can also supply a game result along with the game's schedule entry. If the game list contains more items than there is in SCHEDULE-FORMAT list, the rest is assumed to contain the game result in the format according to GAMERES-FIELDS. If the game result specified in the schedule, the game will be read-only. This obscure feature allows to implement carry-forward games but can also be used for pre-setting game results in the schedule.

tour-name:SCHEDULE-DATE variable should be provided. It is a string that describes the format of the DATE field in game definitions. The format description style used is identical to the one of UNIX strptime(3) C function. For instance, a date "2001-11-12 18:30" would be have format "%Y-%m-%d %H:%M" Please see the corresponding man page for details.

tour-name:SCHEDULE-HOME variable notifies the position of the home team in the game definitions. The following values (integer type) are permitted:

-1	home/away notion has no meaning for this tournament
0	the team in TEAM1 field is the home team
1	the team in TEAM2 field is the home team

7 Scheme modules

A number of C++-defined functions are exported into the `(libtour builtin)` module and can be used by Scheme procedures within a tournament definition.

A tournament definition which is a list of lists of lists ... can be tedious to write. In order to simplify it as well as bring some fun into rules definition a helper module `(libtour common)` is provided.

`(libtour filters)` provides some team and game filters for use in stage definitions.

7.1 builtin module

To use these procedures in your modules, do

```
(use-modules (libtour builtin))
```

unless you use `(libtour common)` which `use`s and `re-export`s them.

t-tid *tour-name stage-id group-id pos* [(libtour builtin) procedure]
Return a list of global team IDs for stage *stage-id*, group *group-id* at position *pos*. If the stage is not initialized (see Section 5.3 [TStage state transitions], page 13), return `#f`. Note that this procedure returns a list rather than a string since more than one team can occupy a position within a group due to a sorting conflict.

t-pos *tour-name stage-id group-id global-team-id* [(libtour builtin) procedure]
Return a pair of positions (low and high) for the team *global-team-id* in the group *group-id* of stage *stage-id*. Note that *global-team-id* is a global team ID.

t-gl-tid *tour-name stage-id local-team-id* [(libtour builtin) procedure]
Return global team ID for a team with local team ID *local-team-id* in stage *stage-id*.

t-lc-tid *tour-name stage-id global-team-id* [(libtour builtin) procedure]
Return local team ID for a team with global team ID *global-team-id* in stage *stage-id*.

t-lc-tid-by-name *tour-name stage-id team-name* [(libtour builtin) procedure]
Return local team ID for a team with name *team-name* in stage *stage-id*.

team-belongs-to-group *tour-name stage-id* [(libtour builtin) procedure]
group-id local-team-id
Return true if the group *group-id* within stage *stage-id* contains team with stage-local team id *local-team-id*.

make-tmp-group *tour-name stage-id group-spec* [(libtour builtin) procedure]
Make a temporary group within the stage *stage-id* according to the group specification *group-spec*. See Section 6.3.4 [Groups definition], page 22, for a complete description of a group specification list. A group created with `*make-tmp-group*` can later be deleted with `*delete-tmp-group*`.

delete-tmp-group *tour-name group-id* [(libtour builtin) procedure]
Delete a temporary group *group-id* within stage *stage-id*. Note that only groups created with `*make-tmp-group*` can be deleted.

get-games *tour-name stage-id group-id only-complete-games* [(libtour builtin) procedure]
Return list of the games for group *group-id* within stage *stage-id*. If *only-complete-games* flag is `#t` omit the games that haven't been played yet. The return list is in the following format:

```
Return list: '(<game> ...)

<game>: (<game-spec> <game-res>)

<game-spec>:
fields for the game as defined in SCHEDULE-FORMAT

<game-res>: if the game result is set, fields for the game result as
defined in GAMERES-FIELDS. If the game result is not set, empty
list.
```

If *only-complete-games* is `#t` only games with a result set will be returned.

See Section 6.5 [Game schedule definition], page 26, for `SCHEDULE-FORMAT` and See Section 6.3.1 [Standings and Game result fields], page 21, for `GAMERES-FIELDS` definitions.

is-complete *tour-name stage-id . group-id* [(libtour builtin) procedure]
Return `#t` if the group *group-id* within the stage *stage-id* is complete. If *group-id* argument is omitted, return `#t` if all groups in stage *stage-id* are complete (the stage is complete).

home-position *tour-name* [(libtour builtin) procedure]
Return an interger for a team's home position as defined in `SCHEDULE-HOME` variable (see Section 6.5 [Game schedule definition], page 26). The return value can be `-1` (no home/away), `0` (home team comes first), or `1` (home team comes second). *tour-name* is the name of the tournament.

time-less *tour-name time-1 time-2* [(libtour builtin) procedure]
This is a "less-than" operation for date/time strings in the format defined in the `SCHEDULE-DATE` variable (see Section 6.5 [Game schedule definition], page 26). It returns `true` if *time-1* is less than *time-2*, and `false` otherwise. *tour-name* is the name of the tournament.

7.2 common module

Make sure to import the `(libtour common)` module before calling the procedures described below.

cmp-assoc-field-p *key alist-1 alist-2* [(libtour common) procedure]
Compare values for the key *key* in two associated lists (*alist-1* and *alist-2*). Return `1` if the value in the first list is greater than the value in the second list, `-1` if less, and `#f` if they are equal. No error checking is done.

cmp-assoc-field-diff-p *key-1 key-2 alist-1 alist-2* [(libtour common) procedure]
Like `cmp-assoc-field-p`, only compares difference of values denoted by keys *key-1* and *key-2* in two associated lists.

cmp-two-teams-p *tour-name stage-id team-id-1 team-id-2 cmp-f game-here-f* [(libtour common) procedure]
Compare personal results between two teams specified by local team IDs *team-id-1* and *team-id-2*. The teams must belong to stage `stage-id`. *cmp-f* is a procedure to compare the teams (see Section 6.3.4.3 [Teams comparison procedure], page 23), and *game-here-f* is a procedure to decide which games should be taken into account (see Section 6.3.4.4 [game-belongs-to-group procedure], page 24).

Return `1`, `-1`, or `#f` if the first teams' result is greater, less, or equal to the second team correspondingly.

`cmp-two-teams-p` will create a temporary group in the stage *stage-id* with only two teams, sort them and compare their position within the group. The temporary group is guaranteed to be deleted in the presence of exceptions.

uniq-tid *tour-name stage-id group-id pos* [(libtour common) procedure]
Return *global* team ID at position *pos* in group *group-id* within stage *stage-id*, or `#f` if more than one team share that position. Will throw error if stage *stage-id* is "uninitialized" (see Section 5.3 [TStage state transitions], page 13). Note that this procedure does not check if stage/group is complete.

winner-tid *tour-name stage-id group-id pos* [(libtour common) procedure]
Just like `uniq-tid` but return `#f` immediately if the group *group-id* is not complete.

map-value->thunk *alist* [(libtour common) procedure]
Replace each value of *alist* associative list with a thunk that returns the value when called. Return the resulting list.

map-tid-spec->thunk *alist . proc* [(libtour common) procedure]
Given associative list *alist* in the form `(("team-id" "stage-id" "group-id" position) ...)` produce new associative list in the form `(("team-id" . <thunk>) ...)` so that execution of the thunk will produce a corresponding global team id for that specification. If procedure *proc* is present, use it, else use `uniq-tid`.

Note: this is a peculiar procedure used to simplify calls to `uniq-tid` etc. There must be a nicer way...

mk-res-lists *spec* [(libtour common) procedure]
Transform the associative list *spec* in the form `((key val1 val2) ...)` into a pair of associative lists `(((key . val1) ...) (key . val2) ...)`. Used in the game interpretation procedures (see Section 6.3.2 [Game results interpretation procedure], page 21).

make-group *#:id group-id #:name group-name #:teams teams-list #:formula group-formula #:cmp-f compare-f #:game-here-f game-belongs-here-f* [(libtour common) procedure]
g-formula-default [(libtour common) fluid]
g-cmp-f-default [(libtour common) fluid]
g-game-here-f-default [(libtour common) fluid]
Make and return a group specification list (see Section 6.3.4 [Groups definition], page 22) given the argument denoted by keywords. It is possible to dynamically bind the last three arguments using Guile fluids so that they can be omitted. For instance (from the '`ihwc2003_rules.scm`'):

```
(with-fluids
 ((g-game-here-f-default 'ANY)
  (g-formula-default '(ROUND 1))
  (g-cmp-f-default cmp-prelim-stage-f))
 (list (make-group #:id "A" #:teams '("SVK" "GER" "UKR" "JPN"))
       (make-group #:id "B" #:teams '("RUS" "USA" "SUI" "DEN"))
       (make-group #:id "C" #:teams '("SWE" "CAN" "LAT" "BLR"))
       (make-group #:id "D" #:teams '("FIN" "CZE" "AUT" "SLO"))))
```

make-stage *#:id stage-id #:name stage-name #:standings-fields standings-fields #:gameres-fields gameres-fields #:res-f res-f #:teams teams-spec #:groups groups-spec* [(libtour common) procedure]
Make and return a stage specification list (see Section 6.3 [Stages definition], page 20). No fluids here.

7.3 filters module

Filter-making procedures are used in creation of filtered groups (see Section 5.6 [Filtered groups], page 15). You have to explicitly list the filter-making procedures you want to make avalable for a stage in the `FILTERS` field of the stage definition (see Section 6.3.6 [Filters definition], page 25).

Make sure to import the `(libtour filters)` module before using the procedures described below.

empty *tname st-id gr-id* [(libtour filters) procedure]
A game filter. If used will make sure the group has no games. Useful for testing. Returns a procedure that takes a game associative list as the sole argument.

at-home *tname st-id gr-id* [(libtour filters) procedure]
A game filter. A group that uses it will only consider home games results. This filter cannot be used if `SCHEDULE-HOME` is `-1` (see Section 6.5 [Game schedule definition], page 26). Returns a procedure that takes a game associative list as the sole argument.

at-away *tname st-id gr-id* [(libtour filters) procedure]
A game filter. A group that uses it will only consider away games results. This filter cannot be used if `SCHEDULE-HOME` is `-1` (see Section 6.5 [Game schedule definition], page 26). Returns a procedure that takes a game associative list as the sole argument.

vs-group *tname st-id gr-id vs-gr-id* [(libtour filters) procedure]
A game filter. A group that uses it will only have games played versus group with ID *vs-gr-id* (`string` type). Returns a procedure that takes a game associative list as the sole argument.

vs-teams *tname st-id gr-id action team-ls* [(libtour filters) procedure]
A game filter. A group that uses it will only consider games that satisfies *action* argument against team stage-local IDs in *team-ls*. *action* is of the `symbol` type and one of

- equals
- !equals
- contains
- !contains

Returns a procedure that takes a game associative list as the sole argument.

time-span *tname st-id gr-id time-1 time-2* [(libtour filters) procedure]
A game filter. A group that uses it will only have games played between *time-1* (inclusive) and *time-2* (exclusive). The time arguments are of the `string` type, and their format should be conform to the `SCHEDULE-DATE` definition (see Section 6.5 [Game schedule definition], page 26). Returns a procedure that takes a game associative list as the sole argument.

game-fld *tname st-id gr-id fld-name action val-ls* [(libtour filters) procedure]
A game filter. A group that uses it will only consider games that satisfies *action* argument towards contents of field *fld-name* (`symbol` type) in *val-ls*. *val-ls* is a `list` of values (`string` type). *action* is of the `symbol` type and one of

- equals
- !equals
- contains
- !contains

Returns a procedure that takes a game associative list as the sole argument.

team-fld *tname st-id gr-id fld-name action val-ls* [(libtour filters) procedure]

A team filter. A group that uses it will only contain teams that satisfy *action* argument towards contents of the team field *fld-name* (`symbol` type) in *val-ls*. *val-ls* is a `list` of values (`string` type). *action* is of the `symbol` type and one of

- equals
- !equals
- contains
- !contains

Returns a procedure that takes a team associative list as the sole argument.

8 libtour API

Header files named '`t_*.hh`' are intended to be used by a program that links `libtour`. These headers are normally installed in the '`$prefix/include`' directory. The following sections describe how to use the `libtour` API. The examples are using STL's `string` class.

8.1 Initializing libtour

First of all, boot guile (the flow control will resume in `innner_main` once Guile is initialized):

```
#include <t_tour.hh>
#include <t_exception.hh>
#include <config.h>

int
main( int argc, char** argv )
{
    scm_boot_guile(argc, argv, inner_main, 0);
    return 0;
}

static void
inner_main( void*, int argc, char** argv )
{
    ...
```

Now you have to set path(s) to your tournament definitions. Use either `T_Tour::appendPath()` to append a single directory:

```
string mydir = ...
T_Tour::appendPath( mydir );
```

or `T_Tour::setPath()` to use several directories:

```
vector<string> mydirs;
...
T_Tour::setPath( mydirs );
```

`libtour` installation contains a directory with several tournament definitions, its location can be retrieved with `libour-config` command line tool.

`libtour` user code should be enclosed in a `try ... catch` block since the `libtour` calls may throw `T_Exception` objects.

```
try {
    T_Tour tour( tour_name ); //parse the tournament definition
    ...
}
catch ( T_Exception& e ) { cerr << e.why() << endl; }
```

It is possible to distinguish between errors caused by invalid tournament definitions (Scheme code) and invalid requests on one side, and internal `libtour` errors on the other. For

details on the exception classes' hierarchy See Section 5.1 [Main classes], page 11, and 't_exception.hh' header.

Inside the `try ... catch` block after the `tour_name` tournament definition is parsed in the `T_Tour` constructor, we can load previously saved results (see Section 5.8 [Saving and restoring state], page 17), or get the latest ones and save them:

```
string saved_results = getSavedResults();
tour.loadResults( saved_results );
...

string latest_results = tour.getResults();
saveResults( latest_results );
```

where `getSavedResults()` and `saveResults()` read and write a result file. And you can, of course, query and modify the tournament state.

8.2 Interacting with the tournament

To make a query (either read-only or modifying) you have to prepare a `T_QuerySpec` object and pass it to the `T_Tour` instance:

```
#include <t_tour.hh>
#include <t_query.hh>

...
    T_Tour tour( tour_name );

    T_QuerySpec q;
    ... //fill in the query

    T_QueryResults res;
    tour.processQuery( q, res );
    ... //process result

    T_QuerySpec q2;
    ... //fill in the insertion request
    tour.processInsert( q2 );
```

`T_Tour::processQuery()` will fill in a `T_QueryResults` object passed to it with the results of the query. `T_Tour::processInsert()`, on the other hand, changes the state of a tournament and returns nothing. Both of these methods will throw a `T_Exception` if anything goes wrong.

The following sections explain how to make a query (see Section 8.3 [Making a query object], page 35) and interpret the result (see Section 8.4 [Processing query result], page 36).

8.3 Making a query object

To communicate with `libtour` the user code has to encaplulate its request into a `T_QuerySpec` object and pass it to either `T_Tour::processQuery()` or `T_Tour::processInsert()` method (see Section 8.2 [Interacting with the tournament], page 34).

There are two parts in a `T_QuerySpec` object:

1. **target** — what to get
2. **options** — how to get it

Both target and options of a query should belong to the `T_QuerySpec::Token` enumeration:

```
class T_QuerySpec
{
    enum Token {
        UNDEFINED=-1,
        ID,              //id
        INFO,            //info
        NAME,            //name
        STAGE,           //stage
        GROUP,           //group
        DEL_GROUP,       //delete usr group
        TEAM,            //team
        TEAM1,           //first team
        TEAM2,           //second team
        GAME,            //game
        UN_GAME,         //undo game
        SCHEDULE,        //schedule
        STANDINGS,       //standings
        S_CONFLICT,      //sorting conflict
        S_CONFLICT_DET,  //sorting conflict details
        S_CONFLICT_SOL,  //solution of a sorting conflict
        POS_RANGE,       //position range
        G_RESULT,        //game result
        COMPLETE,        //complete stage
        UN_COMPLETE,     //uncomplete stage
        TEAM_FIELDS,     //team result fields
        STANDINGS_FIELDS,//team result fields
        GAME_FIELDS,     //game result fields
        SCHEDULE_FIELDS, //schedule fields
        DATE_FORMAT,     //date/time format
        HOME_POSITION,   //position of the home team
        EXCESSIVE_GAMES, //ids of excessive games
        BELONGS,         //game belongs to group
        WINNERS,         //winners
        TEAM_FILTER,     //team filter for making usr groups
        GAME_FILTER,     //game filter for making usr groups
```

```
            FILTER_ARGS      //team or game filter arguments
            MAX_POINTS       //max points in a game
        };
        ...
    };
```

For instance, to query the standings table of the group *A* of the stage *S1* one would do the following:

```
T_QuerySpec qs;
qs.setTarget( T_QuerySpec::STANDINGS );
qs.setOption( T_QuerySpec::STAGE, "S1" );
qs.setOption( T_QuerySpec::GROUP, "A" );

T_QueryResults res;
tour.processQuery( qs, res );
```

Naturally, not all combinations of target and options make sence. Passing invalid queries will result in a `T_RuntimeException` exceptions thrown. See Section 8.5 [Query reference], page 37, for a complete listing of possible queries and their results.

8.4 Processing query result

A `T_QueryResults` instance filled in by the `T_Tour::processQuery()` call contains tabular results of the query. For example, a standings query will build a `T_QueryResults` object representing the table of standings etc. The cells are of the abstract `T_QueryData` type. Columns of the table are guaranteed to contain values of the same type derived from `T_QueryData`. This can be one of the following: `T_QueryStringData`, `T_QueryIntData`, `T_QueryIntRangeData`, or `T_QueryBoolData`. One can treat the cells polymorphically by using `T_QueryData::toString()` virtual method. If, however, it is required to access methods of a particular derived class, you will have to downcast.

The code below shows how to loop through a `T_QueryResults` object:

```
#include <t_query.hh>
...

//get T_QueryResults res

    for ( size_t row = 0; row < res.nRows(); ++row ) {
        for ( size_t col = 0; col < res[row].nCols(); ++col ) {
            cout << res[row][col].toString() << ' ';
        }
        cout << endl;
    }
```

Alternatively, you can get the cell content by its row number and column name by means of `T_QueryResults::value()`:

```
    for ( size_t row = 0; row < res.nRows(); ++row ) {
        cout << res.value( row, "DATE" );
```

Some meta-information (column names mentioned above, their alignment, and the query caption) is stored in a `T_QueryResults` object as well:

```
class T_QueryResults
{
public:

    /* returns vector of column alignments, -1/0/1 values
       suggests left/center/right alignment  */
    const vector<int>& alignment() const;

    /* returns vector of headings of the columns */
    const vector<string>& heading() const;

    /* returns caption for the query results */
    const string& caption() const;
```

Please refer to the implementation of the CLI user application included in the `libtour` distribution for a working example of `T_QueryResults` processing.

8.5 Query reference

Description of each query consists of the following parts:

- `T_QuerySpec` target
- `T_QuerySpec` options
- `T_QueryResults` columns and their types

Target and options specification is used for constructing the proper `T_QuerySpec` object (see Section 8.3 [Making a query object], page 35). Details related to `T_QueryResults` describe what to expect in the returned results (see Section 8.4 [Processing query result], page 36).

Please note that all read-only queries are guaranteed to succeed (provided they are correct) for any stage in any state. For stages in uninitialized state the result will contain fake data (e.g. stage-local team IDs in place of real team names) or empty strings. Refer to the following subsections for details.

8.5.1 Read-only queries

info [read-only query]

Query tournament information

TARGET

- `T_QuerySpec::INFO`

OPTIONS

RESULT

- Fields defined in `<tour-name>::INFO` variable (see Section 6.1 [Defining a tournament module], page 19), *string* type for each field

stages [read-only query]

Query stage information

TARGET

- `T_QuerySpec::STAGE`

OPTIONS

- `T_QuerySpec::STAGE` — stage ID as in `ID` field of the stage definition (see Section 6.3 [Stages definition], page 20), *optional*

RESULT

- `Id` — stage ID as in `ID` field of the stage definition (see Section 6.3 [Stages definition], page 20), *string* type
- `Name` — stage name as in `NAME` field of the stage definition (see Section 6.3 [Stages definition], page 20), *string* type
- `State` — stage state (see Section 5.3 [TStage state transitions], page 13), one of `T_STATE_UNINIT`, `T_STATE_READY`, `T_STATE_COMPLETE`; *string* type

groups [read-only query]

Query group(s) for a stage.

TARGET

- `T_QuerySpec::GROUP`

OPTIONS

- `T_QuerySpec::STAGE` — stage ID as in `ID` field of the stage definition (see Section 6.3 [Stages definition], page 20)
- `T_QuerySpec::GROUP` — group ID as in `ID` field of the group definition (see Section 6.3.4 [Groups definition], page 22), *optional*

RESULT

- `Id` — group ID as in `ID` field of the group definition (see Section 6.3.4 [Groups definition], page 22), *string* type
- `Name` — group name as in `NAME` field of the group definition (see Section 6.3.4 [Groups definition], page 22), *string* type
- `Description` — optional group description (see Section 6.3.4 [Groups definition], page 22), *string* type
- `State` — group state one of `T_STATE_UNINIT`, `T_STATE_READY`, `T_STATE_COMPLETE`; (see Section 5.4 [Game results and formula], page 14), *string* type
- `Schedule` — whether the group schedule is sufficient, *bool* type; it is undefined for groups with no formula (see Section 6.3.4.2 [Group formula definition], page 23) or for any group in an uninitialized state
- `Summary` — whether the group has a formula (see Section 6.3.4.2 [Group formula definition], page 23), *bool* type
- `Filtered` — whether the group was created by the user as a filtered group (see Section 5.6 [Filtered groups], page 15), *bool* type

teams [read-only query]

Query teams of a group of a stage.

TARGET

- `T_QuerySpec::TEAM`

OPTIONS

- `T_QuerySpec::STAGE` — stage ID as in `ID` field of the stage definition (see Section 6.3 [Stages definition], page 20)
- `T_QuerySpec::GROUP` — group ID as in `ID` field of the group definition (see Section 6.3.4 [Groups definition], page 22)

RESULT

- `GbId` – global team ID as defined in the Scheme data (see Section 6.2 [Teams definition], page 20). Empty string if the stage is uninitialized.
- `LCId` – stage-local team ID (see Chapter 2 [Structure of a tournament], page 3)
- `TEAM-FORMAT` fields (see Section 6.2 [Teams definition], page 20), *string* type EACH. If the stage is uninitialized `NAME` field will contain the stage-local team ID surrounded by angle brackets, and the rest of the `TEAM-FORMAT` fields will be empty.

schedule [read-only query]

Query game schedule for a stage. Filter on group ID and/or game IDs if provided.

TARGET

- `T_QuerySpec::SCHEDULE`

OPTIONS

- `T_QuerySpec::STAGE` — stage ID as in `ID` field of the stage definition (see Section 6.3 [Stages definition], page 20)
- `T_QuerySpec::GROUP` — group ID as in `ID` field of the group definition (see Section 6.3.4 [Groups definition], page 22) *optional*
- `T_QuerySpec::GAME` — space-separated list of game IDs as in `ID` field of the game definition (see Section 6.5 [Game schedule definition], page 26), *optional*

RESULT

- `SCHEDULE-FORMAT` fields (see Section 6.5 [Game schedule definition], page 26), *string* type for each field. If the stage is uninitialized `TEAM1` and `TEAM2` fields will contain the stage-local team ID surrounded by angle brackets
- `GAMERES-FIELDS` fields (see Section 6.3.1 [Standings and Game result fields], page 21), types as defined in `GAMERES-FIELDS`. Empty if game has no result
- `p1` — points the first team earned, *int* type. Empty if game has no result
- `p2` — points the second team earned, *int* type. Empty if game has no result
- `ro` — read-only flag, *bool* type

standings [read-only query]

Query table of standings for a group within a stage.

TARGET

- `T_QuerySpec::STANDINGS`

OPTIONS

- `T_QuerySpec::STAGE` — stage ID as in `ID` field of the stage definition (see Section 6.3 [Stages definition], page 20)
- `T_QuerySpec::GROUP` — group ID as in `ID` field of the group definition (see Section 6.3.4 [Groups definition], page 22)

RESULT

- `Pos` — team position, *int range* type
- `Team` — team name as in `NAME` field of teams definition (see Section 6.2 [Teams definition], page 20), *string* type
- `STANDINGS-FIELDS` fields (see Section 6.3 [Stages definition], page 20), *int* type

sorting-conflicts [read-only query]

Query sorting conflicts for a group.

TARGET

- `T_QuerySpec::S_CONFLICT`

OPTIONS

- `T_QuerySpec::STAGE` — stage ID as in `ID` field of the stage definition (see Section 6.3 [Stages definition], page 20)
- `T_QuerySpec::GROUP` — group ID as in `ID` field of the group definition (see Section 6.3.4 [Groups definition], page 22)

RESULT

- `Pos` — team position, *int range* type
- `Resolved` — whether this conflict is resolved, *bool* type

sorting-conflict-details [read-only query]

Query details of a sorting conflicts for a group.

TARGET

- `T_QuerySpec::S_CONFLICT_DET`

OPTIONS

- `T_QuerySpec::STAGE` — stage ID as in `ID` field of the stage definition (see Section 6.3 [Stages definition], page 20)
- `T_QuerySpec::GROUP` — group ID as in `ID` field of the group definition (see Section 6.3.4 [Groups definition], page 22) `T_QuerySpec::POS_RANGE` — a string representing position range as returned by the read-only `T_QuerySpec::S_CONFLICT` query

RESULT

- `Team` — team name, *string* type
- `Pos` — `-1` if this conflict is not resolved, otherwise `0 or greater` to notify the relative team position within this conflict; *int* type

excessive *games* [read-only query]

Query excessive game IDs for [a group within] a stage

TARGET

- `T_QuerySpec::EXCESSIVE_GAMES`

OPTIONS

- `T_QuerySpec::STAGE` — stage ID as in `ID` field of the stage definition (see Section 6.3 [Stages definition], page 20)
- `T_QuerySpec::GROUP` — group ID as in `ID` field of the group definition (see Section 6.3.4 [Groups definition], page 22), *optional*

RESULT

- `Id` — game ID as in `ID` field of the schedule definition (see Section 6.5 [Game schedule definition], page 26), *string* type

groups-that-contain-a-game [read-only query]

Query group IDs within a stage that contain a game with given ID

TARGET

- `T_QuerySpec::BELONGS`

OPTIONS

- `T_QuerySpec::STAGE` — stage ID as in `ID` field of the stage definition (see Section 6.3 [Stages definition], page 20)
- `T_QuerySpec::GAME` – game ID as in `ID` field of the schedule definition (see Section 6.5 [Game schedule definition], page 26)

RESULT

- `Id` — group ID as in `ID` field of the group definition (see Section 6.3.4 [Groups definition], page 22), *string* type

team-fields [read-only query]

Query team fields.

TARGET

- `T_QuerySpec::TEAM_FIELDS`

OPTIONS

RESULT

- `Value` — names of `TEAM-FORMAT` fields (see Section 6.2 [Teams definition], page 20), *string* type

standings-fields [read-only query]

Query standings fields for a stage.

TARGET

- `T_QuerySpec::STANDINGS_FIELDS`

OPTIONS

- `T_QuerySpec::STAGE` — stage ID as in `ID` field of the stage definition (see Section 6.3 [Stages definition], page 20)

RESULT

- `Value` — names of `STANDINGS-FIELDS` fields (see Section 6.3.1 [Standings and Game result fields], page 21), *string* type for each field

game-result-fields [read-only query]

Query game result fields for a stage.

TARGET

- `T_QuerySpec::GAME_FIELDS`

OPTIONS

- `T_QuerySpec::STAGE` — stage ID as in `ID` field of the stage definition (see Section 6.3 [Stages definition], page 20)

RESULT

- `Value` — names of `GAMERES-FIELDS` fields (see Section 6.3.1 [Standings and Game result fields], page 21), *string* type
- `Type` — types of `GAMERES-FIELDS` fields (see Section 6.3.1 [Standings and Game result fields], page 21), *string* type

schedule-fields [read-only query]

Query game schedule fields for a stage.

TARGET

- `T_QuerySpec::SCHEDULE_FIELDS`

OPTIONS

RESULT

- `Value` — names of `SCHEDULE-FORMAT` fields (see Section 6.5 [Game schedule definition], page 26), *string* type

date-format [read-only query]

Query date format used in the schedule definition.

TARGET

- `T_QuerySpec::DATE_FORMAT`

OPTIONS

RESULT

- `Value` — date format as in `strptime(3)` C library function (see Section 6.5 [Game schedule definition], page 26), *string* type

winners [read-only query]

Query winners of the tournament. Will *not* throw exception if winners are not known yet.

TARGET

- `T_QuerySpec::WINNERS`

OPTIONS

RESULT

- `Winner` — winner names as in keys of the winners definition (see Section 6.4 [Winners definition], page 25), *string* type
- `Name` — team name as in `NAME` field of teams definition (see Section 6.2 [Teams definition], page 20), *string* type

home-team-position [read-only query]

Query value of the `SCHEDULE-HOME` definition (see Section 6.5 [Game schedule definition], page 26)

TARGET

- `T_QuerySpec::HOME_POSITION`

OPTIONS

RESULT

- `Value` — value for the home team position, *int* type

team-filters [read-only query]

Query team fiters available for a stage (see Section 5.6 [Filtered groups], page 15).

TARGET

- `T_QuerySpec::TEAM_FILTER`

OPTIONS

- `T_QuerySpec::STAGE` — stage ID as in `ID` field of the stage definition (see Section 6.3 [Stages definition], page 20)

RESULT

- `Name` — filter name which corresponds to name of the procedures in the `TEAM-FILTERS` definition (see Section 6.3.6 [Filters definition], page 25), *string* type
- `Description` — documentation of the filter-making procedure, *string* type

game-filters [read-only query]

Query game fiters available for a stage (see Section 5.6 [Filtered groups], page 15).

TARGET

- `T_QuerySpec::GAME_FILTER`

OPTIONS

- `T_QuerySpec::STAGE` — stage ID as in `ID` field of the stage definition (see Section 6.3 [Stages definition], page 20)

RESULT

- `Name` — filter name which corresponds to name of the procedures in the `GAME-FILTERS` definition (see Section 6.3.6 [Filters definition], page 25), *string* type
- `Description` — documentation of the filter-making procedure, *string* type

filter-arguments [read-only query]

Query arguments for a filter and their types (see Section 5.6 [Filtered groups], page 15).

TARGET

- `T_QuerySpec::FILTER_ARGS`

OPTIONS

- `T_QuerySpec::STAGE` — stage ID as in `ID` field of the stage definition (see Section 6.3 [Stages definition], page 20)
- `T_QuerySpec::TEAM_FILTER` — name of a team filter available for this stage (see Section 6.3.6 [Filters definition], page 25)
- `T_QuerySpec::GAME_FILTER` — name of a game filter available for this stage (see Section 6.3.6 [Filters definition], page 25)

Note that `TEAM_FILTER` and `GAME_FILTER` options are *mutually exclusive*.

RESULT

- `Name` — description of the argument, *string* type
- `Type` — type of the argument, *string* type (see Section 5.6 [Filtered groups], page 15), *string* type

max-game-points [read-only query]

Query maximum points a team can get in a game for a stage.

TARGET

- `T_QuerySpec::MAX_POINTS`

OPTIONS

- `T_QuerySpec::STAGE` — stage ID as in `ID` field of the stage definition (see Section 6.3 [Stages definition], page 20)

RESULT

- `Value` — maximum number of points in a game, *int* type; or a uninitialized value if not defined in rules

8.5.2 Modifying queries

insert-game-result [modifying query]

Insert or replace a game result.

TARGET

- `T_QuerySpec::GAME`

OPTIONS

- `T_QuerySpec::GAME` – game ID as in `ID` field of the schedule definition (see Section 6.5 [Game schedule definition], page 26)
- `T_QuerySpec::G_RESULT` – a string representing a game result; values corresponding to `GAMERES-FIELDS` fields (see Section 6.3.1 [Standings and Game result fields], page 21) separated by spaces
- `T_QuerySpec::STAGE` — stage ID as in `ID` field of the stage definition (see Section 6.3 [Stages definition], page 20)

delete-game-result [modifying query]

Delete a game result.

TARGET

- `T_QuerySpec::UN_GAME`

OPTIONS

- `T_QuerySpec::GAME` – game ID as in `ID` field of the schedule definition (see Section 6.5 [Game schedule definition], page 26)
- `T_QuerySpec::STAGE` — stage ID as in `ID` field of the stage definition (see Section 6.3 [Stages definition], page 20)

compete-stage [modifying query]

Set a stage complete.

TARGET

- `T_QuerySpec::COMPLETE`

OPTIONS

- `T_QuerySpec::STAGE` — stage ID as in `ID` field of the stage definition (see Section 6.3 [Stages definition], page 20)

uncompete-stage [modifying query]

Set a stage uncomplete. This will invalidate any ready or complete following stages (see Section 5.3 [TStage state transitions], page 13).

TARGET

- `T_QuerySpec::UN_COMPLETE`

OPTIONS

- `T_QuerySpec::STAGE` — stage ID as in `ID` field of the stage definition (see Section 6.3 [Stages definition], page 20)

resolve-sorting-conflict [modifying query]

Resolve a sorting conflict. The stage has to be in the “ready” state (see Section 5.3 [TStage state transitions], page 13).

TARGET

- `T_QuerySpec::S_CONFLICT`

OPTIONS

- `T_QuerySpec::STAGE` — stage ID as in `ID` field of the stage definition (see Section 6.3 [Stages definition], page 20)
- `T_QuerySpec::GROUP` — group ID as in `ID` field of the group definition (see Section 6.3.4 [Groups definition], page 22)
- `T_QuerySpec::POS_RANGE` — a string representing position range as returned by the read-only `T_QuerySpec::S_CONFLICT` query
- `T_QuerySpec::S_CONFLICT_SOL` — list of team names (in double quotes) in the order they should appear in the table of standings, an empty string if resolution of this sorting conflict should be reset

make-filtered-group [modifying query]

Make a filtered group. The stage has to be in the "ready" or "complete" state (see Section 5.3 [TStage state transitions], page 13).

TARGET

- `T_QuerySpec::GROUP`

OPTIONS

- `T_QuerySpec::STAGE` — stage ID as in `ID` field of the stage definition (see Section 6.3 [Stages definition], page 20)
- `T_QuerySpec::GROUP` — base group ID as in `ID` field of the group definition (see Section 6.3.4 [Groups definition], page 22)
- `T_QuerySpec::ID` — new group ID, should be unique within this stage
- `T_QuerySpec::NAME` — new group name, *optional*
- `T_QuerySpec::TEAM_FILTER` — a team filter expression; a *string* that contains white space delimited tokens: the first is the filter name and the following are the arguments for the corresponding filter-making procedure (see Section 5.6 [Filtered groups], page 15).
- `T_QuerySpec::GAME_FILTER` — a game filter expression; a *string* that contains white space delimited tokens: the first is the filter name and the following are the arguments for the corresponding filter-making procedure (see Section 5.6 [Filtered groups], page 15).

Some examples of using this query can be found in See Appendix A [CLI client sample session], page 47.

delete-filtered-group [modifying query]

Delete a filtered group (see Section 5.6 [Filtered groups], page 15).

TARGET

- `T_QuerySpec::DEL_GROUP`

OPTIONS

- `T_QuerySpec::STAGE` — stage ID as in `ID` field of the stage definition (see Section 6.3 [Stages definition], page 20)
- `T_QuerySpec::GROUP` – group ID to be deleted. Note that this group must have been created with `make-filtered-group` query

Appendix A CLI client sample session

Start the application for NHL 2002 tournament

```
$ tourcli nhl2002
Loading in 0.118012 sec
```

Load results

```
T0ur>> l /usr/local/share/libtour/data/nhl2002_res.txt
Done in 0.382751 sec
```

Get help on the read-only queries

```
T0ur>> h q
Queries, start with ‘q’:
i
- print tour info
s [-s <stage-id>]
- print stage info
g -s <stage-id> [-g <group-id>]
- print groups for stage <stage-id>
t -s <stage-id> [-g <group-id>]
- print teams for stage <stage-id> [group <group-id>]
k -s <stage-id> [-g <group-id>] [-m <game-id-list>
- print schedule for stage <stage-id> [group <group-id>]
p -s <stage-id> -g <group-id>
- print standings for stage <stage-id> group <group-id>
sc -s <stage-id> -g <group-id>
- print sorting conflicts for stage <stage-id> group <group-id>
scd -s <stage-id> -g <group-id> -pos <position-range>
- print details for a sorting conflict for team positions
  <position-range> in stage <stage-id> group <group-id>
x -s <stage-id> [-g <group-id>]
- print excessive game ids for stage <stage-id> [group <group-id>]
b -s <stage-id> -m <game-id>
- print groups that contain game <game-id> for stage <stage-id>
tf -s <stage-id>
- print team fields for stage <stage-id>
gf -s <stage-id>
- print game fields for stage <stage-id>
sf
- print schedule fields
df
- print date/time format used in schedule definitions
hp
- print position of the home team
w
- print winners
ft -s <stage-id>
- print available team filters for custom groups
```

```
fg -s <stage-id>
- print available game filters for custom groups
fa -s <stage-id> -(ft|fg) <filter name>
- print required arguments for a team or game filter
mp -s <stage-id>
- print maximum point in game for stage <stage-id>

Done in 0.00415516 sec
```

Print the stages

```
T0ur>> q s
-----------------------------------
         Stage Information
--- ------------------------ ------
Id  Name                     State
--- ------------------------ ------
S1  Regular Season           c
S2  Conference Quater Finals c
S3  Conference Semi Finals   c
S4  Conference Finals        c
S5  Stanley Cup Games        c
-----------------------------------
Done in 0.00150084 sec
```

Print groups of stage “S1”

```
T0ur>> q g -s S1
-------------------------------------------
         Stage S1: Group Information
----- ------------------ ------------ ------
Id    Name               Description  State
----- ------------------ ------------ ------
East  Eastern Conference              c
West  Western Conference              c
AT    Atlantic Division               c
NE    Northeast Division              c
SE    Southeast Division              c
CT    Central Division                c
NW    Northwest Division              c
PC    Pacific Division                c
ALL   All Teams                       c
-------------------------------------------
Done in 0.00175095 sec
```

Print teams of group “PC” in stage “S1”

```
T0ur>> q t -s S1 -g PC
------------------------------------------------------------
            Regular Season (S1), Group PC, Teams
----- ----------- ------------ -------------------- --------
```

```
GbId  LcId        NAME         ORIGIN                COUNTRY
----- ----------- ------------ --------------------- --------
ANA   Anaheim     Anaheim      Anaheim Mighty Ducks  USA
DAL   Dallas      Dallas       Dallas Stars          USA
LA    LosAngeles  Los Angeles  Los Angeles Kings     USA
PHO   Phoenix     Phoenix      Phoenix Coyotes       USA
SJ    SanJose     San Jose     San Jose Sharks       USA
-------------------------------------------------------------
Done in 0.00125909 sec
```

Print table of standings of group “PC” in stage “S1”

```
TOur>> q p -s S1 -g PC
-------------------------------------------------------------------
                  Regular Season (S1), Group PC
---- ------------ ------ ----- ------ ------- ---- ---- ---- -------
Pos  Team          GAMES  WINS  DRAWS  LOSSES  OTL   GF   GA  POINTS
---- ------------ ------ ----- ------ ------- ---- ---- ---- -------
1    San Jose         82    44      8      27    3  248  199      99
2    Phoenix          82    40      9      27    6  228  210      95
3    Los Angeles      82    40     11      27    4  214  190      95
4    Dallas           82    36     13      28    5  215  213      90
5    Anaheim          82    29      8      42    3  175  198      69
-------------------------------------------------------------------
Done in 0.00162506 sec
```

Print team result fields for stage “S1” (see the table of standings above)

```
TOur>> q tf -s S1
-------
Stage Regular Season (S1), Team Fields
-------
Value
-------
GAMES
WINS
DRAWS
LOSSES
OTL
GF
GA
POINTS
-------
Done in 0.000512123 sec
```

Print groups that contain game “1000” within stage “S1”

```
TOur>> q b -s S1 -m 1000
------
Stage Regular Season (S1), groups that contain game 1000
------
```

```
Group
------
East
West
NE
PC
ALL
------
Done in 0.000503778 sec
```

Print winners

```
TOur>> q w
--------------------------------
             Winners
---------------------- ---------
Winner                 Team
---------------------- ---------
Stanley Cup Winner     Detroit
East Conference Winner  Carolina
West Conference Winner  Detroit
--------------------------------
Done in 0.00117803 sec
```

Print help on modifying queries

```
TOur>> h i
Inserts, start with 'i':
m -m <game-id> -r "<game-res>" -s <stage-id>
- insert result <game-res> for game <game-id>
d -m <game-id> -s <stage-id>
- delete result of game <game-id>
c -s <stage-id>
- set stage <stage-id> to be complete
uc -s <stage-id>
- set stage <stage-id> to be incomplete (invalidates next stage(s)!)
g -s <stage-id> -g <group-id> -id <new-group-id> [-n <new-group-name>] \
[-ft <team-filter>] [-fg <game-filter>]
- create a group with ID <new-group-id> and name
  <new-group-name> within stage <stage-id> after group
  <group-id> using <team-filter> and/or <game-filter>
dg -s <stage-id> -g <group-id>
- delete a group with ID <group-id> within stage <stage-id>
sc -s <stage-id> -g <group-id> -pos <position-range> -sol<team list>
- resolve sorting conflict between teams in <position-range>
  in group <group-id> of stage <stage-id>
Done in 0.00304008 sec
```

Un-complete stage "S5"

```
TOur>> i uc -s S4
```

```
Done in 0.000222921 sec
```

Print team positions in group “ALL” of stage “S4”

```
T0ur>> q p -s S4 -g ALL
-----------------------------------
 Conference Finals (S4), Group ALL
---- --------- ------ --- --- -----
Pos  Team       GAMES  GF  GA  WINS
---- --------- ------ --- --- -----
1-2  Detroit        7  22  13     4
1-2  Carolina       6  10   6     4
3    Colorado       7  13  22     3
4    Toronto        6   6  10     2
-----------------------------------
Done in 0.0014708 sec
```

Print sorting conflicts of group “ALL” in stage “S4”

```
T0ur>> q sc -s S4 -g ALL
--------------
Stage Conference Finals (S4), group ALL, Sorting Conflicts
---- ---------
Pos  Resolved
---- ---------
1-2  -
--------------
Done in 0.000473022 sec
```

Show details for this sorting conflict

```
T0ur>> q scd -s S4 -g ALL -pos 1-2
--------------
Stage Conference Finals (S4), group ALL, Sorting Conflict Details
--------- ----
Team      Pos
--------- ----
Carolina  -1
Detroit   -1
--------------
Done in 0.000473022 sec
```

Resolve the sorting conflict between teams “Carolina” and “Detroit” in group “ALL” within stage “S4”, give preference to “Carolina”

```
T0ur>> i sc -s S4 -g ALL -pos 1-2 -sol "\"Carolina\" \"Detroit\""
Done in 0.000316858 sec
```

Verify that the sorting conflict was resolved

```
T0ur>> q p -s S4 -g ALL
-----------------------------------
 Conference Finals (S4), Group ALL
---- --------- ------ --- --- -----
Pos  Team       GAMES  GF  GA  WINS
```

```
---- --------- ------ --- --- -----
1    Carolina       6  10   6     4
2    Detroit        7  22  13     4
3    Colorado       7  13  22     3
4    Toronto        6   6  10     2
---------------------------------
Done in 0.00136495 sec
```

Print game schedule of group “E” in stage “S4”

```
T0ur>> q k -s S4 -g E
-----------------------------------------------------------------------
              Conference Finals (S4), Group E, Games
----- --------------- --------- --------- --- --- --- --- --- ---
ID    DATE            TEAM1     TEAM2      G1  G2  OT  p1  p2  ro
----- --------------- --------- --------- --- --- --- --- --- ---
1315  16.05.02 19:00  Toronto   Carolina    2   1   -   1   0   -
1316  19.05.02 19:00  Toronto   Carolina    1   2   +   0   1   -
1317  21.05.02 19:00  Carolina  Toronto     2   1   +   1   0   -
1318  23.05.02 19:00  Carolina  Toronto     3   0   -   1   0   -
1319  25.05.02 19:00  Toronto   Carolina    1   0   -   1   0   -
1320  28.05.02 19:00  Carolina  Toronto     2   1   +   1   0   -
1321  30.05.02 19:00  Toronto   Carolina                        -
-----------------------------------------------------------------------
Done in 0.00203395 sec
```

Print excessive games in stage “S4”

```
T0ur>> q x -s S4
-----
Stage Conference Finals (S4),  Excessive games
-----
   Id
-----
 1321
-----
Done in 0.0010891 sec
```

Delete result of game “1320” withing stage “S4”

```
T0ur>> i d -m 1320 -s S4
Done in 0.000285149 sec
```

Print game result fields for stage “S4”

```
T0ur>> q gf -s S4
------------
Stage Conference Finals (S4), Game Fields
------ -----
Value  Type
------ -----
G1     int
G2     int
```

```
OT     bool
------------
Done in 0.000478983 sec
```

Insert result of game “1320” in stage “S4”, result string should correspond to the game result fields for this stage (as in the previous example)

```
TOur>> i m -m 1320 -s S4 -r "2 1 1"
Done in 0.000383139 sec
```

Print available game filters for stage “S1”

```
TOur>> q fg -s S1
-------------------------------------------------
                  Game filters
---------- ------------------------------------
Name       Description
---------- ------------------------------------
at-away    away games
at-home    home games
game-fld   Filter games based on a game field value
time-span  Filter games based on a time span
vs-group   games of a group vs another group
vs-teams   games of a group vs given teams
-------------------------------------------------
Done in 0.00111818 sec
```

Print arguments for the game filter “vs-group” in stage “S1”

```
TOur>> q fa -fg vs-group -s S1
-----------------
Arguments for game filter vs-group
--------- -------
Name     Type
--------- -------
group id  string
-----------------
Done in 0.000692129 sec
```

Create a group “East vs West” with ID “zzz” based on group “East” of stage “S1”, give the argument “West” to the “vs-group” filter

```
TOur>> i g -s S1 -g East -id zzz -n "East vs West" \
       -fg "vs-group \"West\""
Done in 0.0687809 sec
```

Print groups for stage “S1”

```
TOur>> q g -s S1
-------------------------------------------------------------------
                  Stage S1: Group Information
----- ------------------ --------------------------------- ------
Id    Name               Description                       State
----- ------------------ --------------------------------- ------
East  Eastern Conference                                   c
```

```
zzz   East vs West        games by group zzz vs group West  c
West  Western Conference                                    c
AT    Atlantic Division                                     c
NE    Northeast Division                                    c
SE    Southeast Division                                    c
CT    Central Division                                      c
NW    Northwest Division                                    c
PC    Pacific Division                                      c
ALL   All Teams                                             c
-----------------------------------------------------------------
Done in 0.00174999 sec
```

Print standings for the group we just created

```
TOur>> q p -s S1 -g zzz
-------------------------------------------------------------------------
                    Regular Season (S1), Group zzz
---- --------------- ------ ----- ------ ------- ---- --- --- -------
Pos  Team             GAMES  WINS  DRAWS  LOSSES  OTL  GF  GA  POINTS
---- --------------- ------ ----- ------ ------- ---- --- --- -------
1    Toronto            22    14      1       6    1  77  65      30
2    Tampa Bay          22    12      2       7    1  48  37      27
3    Boston             22    11      3       6    2  56  46      27
4    N.Y. Islanders     22    11      1       9    1  60  62      24
5-6  Ottawa             22    10      2       8    2  67  60      24
5-6  Carolina           22    10      4       8    0  60  60      24
7    New Jersey         22    10      2       9    1  56  48      23
8    Washington         22    10      2      10    0  57  63      22
9    Philadelphia       22     9      3       9    1  62  57      22
10   Montreal           22     9      3      10    0  54  60      21
11   Buffalo            22     7      2      13    0  55  64      16
12   Florida            22     6      3      12    1  42  64      16
13   N.Y. Rangers       22     6      1      14    1  47  77      14
14   Pittsburgh         22     4      0      17    1  43  83       9
15   Atlanta            22     3      1      17    1  35  77       8
-------------------------------------------------------------------------
Done in 0.00483418 sec
```

Delete the group we created

```
TOur>> i dg -g zzz -s S1
Done in 0.00029397 sec
```

Create a group with the default name and ID “zzz” based on group “ALL” of stage “S1” so that it only contains Canadian teams and games played in November 2001

```
TOur>> i g -s S1 -g ALL -id zzz -ft "team-fld COUNTRY equals \
      (\"Canada\")" -fg "time-span \"1.11.01 0:0\" \"1.12.01 0:0\""
Done in 0.0682631 sec
```

Print standings for the group we just created

```
TOur>> q p -s S1 -g zzz
```

```
------------------------------------------------------------------
                Regular Season (S1), Group zzz
---- ---------- ------ ----- ------ ------- ---- --- --- -------
Pos  Team        GAMES  WINS  DRAWS  LOSSES  OTL  GF  GA  POINTS
---- ---------- ------ ----- ------ ------- ---- --- --- -------
1-3  Toronto        14     7      1       4    2  33  30      17
1-3  Montreal       13     7      2       3    1  34  31      17
1-3  Edmonton       13     7      2       3    1  30  18      17
4    Vancouver      16     7      2       7    0  38  36      16
5    Ottawa         10     6      2       2    0  33  23      14
6    Calgary        13     5      4       4    0  36  35      14
------------------------------------------------------------------
Done in 0.00299215 sec
```

Concept index

U

W

Scheme procedure index

Query index