The Valet Score:

The Library (DLL)

November 2015
Søren Hein
soren.hein@gmail.com

Table of Contents

1 Description

The Valet program can be compiled and run in stand-alone form. It requires tournament data to be available in a certain format, and it produces entire rankings.

The Valet calculation as such can also be used as a library: A DLL on Windows and a static library on Linux and Apple. The library only operates on data for a single board at a time. It is the user's responsibility to keep track of players, pairs and results in order to aggregate the results into overall rankings.

The library interface is chosen to be similar in spirit to DDS, the leading open-source double-dummy solver (of which I am a co-author). It is not a very exciting interface, but the DDS interface has been used from a lot of languages on a lot of systems, so I hope that the interface will not be a limiting factor for Valet either.

The code base in the `src` directory supports the stand-alone program, the library and a simple driver program. If you want to see how to interface to the library from C or C++, look here; at least for these languages you should `#include "valet.h"`. The driver example also shows error handling.

Please see `valet.h` for the exact definition of the functions and data structures.

2 Functions

2.1 VALETINIT

Arguments: None.
Return value: `void`.

`ValetInit()` initializes scoring tables. It is called automatically on systems that support this, and this includes all the systems I've tested on. So it is often not necessary to call `ValetInit()`. If you get results with all-zero scores, it is a sign that you should try calling this function before you do anything else with the library.

2.2 VALETSETCONTROL

Arguments: `struct ControlType * control`
Return value: `int` (see error codes).

`ValetSetControl()` contains the options available to the user. These are a subset of the options available in the full Valet program, as many of the Valet options only make sense in the context of overall scoring.

The `ControlType` struct contains `valet`, `leadFlag` and `datumHardRounding`. If you don't call this function, default options will be used (`VALET_SCORING_IAF`, `false` and `false`, respectively).

The field valet must be one of 0, 1 or 2. You can use the `#define`'s in `valet.h`, that is, `VALET_SCORING_DATUM`, `VALET_SCORING_IAF` and `VALET_SCORING_MP`.

The `leadFlag` is a boolean. Use `true` if you want defense scores for the lead separately.

2.3 VALETCLEAR

Arguments: None.
Return value: `void`.

`ValetClear()` clears both the hand results and the internal pair list. There is no strong need to do this rather than calling only `ValetClearHand()`, but on the other hand the pair list will keep growing if you use the same instance of the library a lot. So you might at least call it once per tournament.

2.4 VALETCLEARHAND

Arguments: None.
Return value: `void`.

`ValetClearHand()` clears the hand results that are stored in the library. You **must** call either this function or `ValetClear()` at the beginning of each hand.

2.5 VALETSETBOARDNUMBER

Arguments: `unsigned no`.
Return value: `int` (see error codes).

`ValetSetBoardNumber()` sets the board number (which must be at least 1). It **must** be called at the beginning of each hand, after `ValetClearHand()`.

The library derives the vulnerability from the board number according to normal bridge rules.

2.6 VALETADDBYLINE

Arguments: `char line[VALET_INPUT_MAX_LENGTH]`.
Return value: `int` (see error codes).

There are three functions for getting individual results into the library, and this is one of them. It is the one that is most like the stand-alone Valet program. A line of characters in the same format as the scores file for valet may be given (even though one might have chosen to simplify the format a bit for the library). The explanation is repeated here.

Lines must be of the form 'a|b|c|…'. There must be exactly either 9 or 10 fields in each line, so 8 or 9 vertical bars. It is acceptable to mix such lines, so there can be lines of both types in a single hand. The meaning of the fields is as follows, in order:

- Round number. An integer, minimum value 1. This is only provided for compatibility with the Valet program and is not used in the library.
- Board number. An integer, minimum value 1. The number must be the same as the one that was set with `ValetAddBoardNumber()`.
- The tag for the North player. A tag is a relatively short sequence of characters (`VALET_TAG_MAX_LENGTH` characters or less, including the null termination). It may be an integer, a BBO handle or some other string.
- The tag for the East player.
- The tag for the South player.
- The tag for the West player.
- The contract, which must be exactly one of the following: P, 1C, 1CX, 1CXX, 1D, 1DX, 1DXX, … 7N, 7NX, 7NXX.
- The declarer, which must be exactly one of N, E, S, W.
- The number of tricks taken, which must be 0, 1, …, 13.
- Optionally the opening lead, which must consist of a suit (S, H, D, C) followed by a card (2, 3, …, 9, T, J, Q, K, A).

2.7 VALETADDBYTAG

Arguments: `struct PlayersTagType * players, struct InputResultType * input.`
Return value: `int` (see error codes).

There are three functions for getting individual results into the library, and this is one of them. If you use this one, you should also use `ValetGetNextScoreByTag()` to get Valet scores back out of the library.

The `players` struct contains the four name tags. These have the same definition and constraints as in `ValetAddByLine()` above. The tags may also be integers, in which case they are still treated as strings internally in the library. The fields of `players` are named `north, east, south` and `west`.

The `input` struct contains seven fields that specify a particular result: `level, denom, multiplier, declarer, tricks, leadDenom` and `leadRank`. All these are of type `unsigned`.

The `level` is 0 .. 7. If it is 0, then the hand was passed out, and the other fields are ignored (but they still need to be syntactically correct).

The `denom` has values 0 .. 4 (in the order spades, hearts, diamonds, clubs, notrump). You may not like the order, but at least it is consistent with the DDS double-dummy solver… I suggest using the constants defined in `valet.h`, so `VALET_SPADES, VALET_HEARTS, VALET_SPADES, VALET_CLUBS, VALET_NOTRUMP`.

The `multiplier` may be 0, 1 or 2. I suggest using the constants defined in `valet.h`, so `VALET_UNDOUBLED, VALET_DOUBLED, VALET_REDOUBLED`.

The `declarer` is 0 .. 3. I suggest using the constants defined in `valet.h`, so `VALET_NORTH, VALET_EAST, VALET_SOUTH, VALET_WEST`.

The `tricks` value is 0 .. 13.

The `leadDenom` uses the same encoding as the `denom`. If the `leadFlag` control option is set to false, then the lead is ignored (but it must still be syntactically correct).

The `leadRank` is 2..14. If the `leadFlag` control option is set to false, then the lead is ignored (but it must still be syntactically correct).

2.8 VALETADDBYNUMBER

Arguments: `struct PlayersNumberType * players, struct InputResultType * input.`
Return value: `int` (see error codes).

There are three functions for getting individual results into the library, and this is one of them. This one is completely analogous to `ValetAddByTag()`, except that the player fields are given not as short strings but as unsigned integers (with value at least 1). If you use this one, you should also use `ValetGetNextScoreByNumber()` to get Valet scores back out of the library.

2.9 VALETCALCULATE

Arguments: None.
Return value: `int` (see error codes).

Once you have passed in all of the results on a given hand, you should call `ValetCalculate()` once. This doesn't return any results, but it makes it possible to get the results with the following functions.

2.10 VALETGETNEXTSCOREBYTAG

Arguments: `struct PositionsTagType * players, struct OutputResultType * output.`
Return value: `bool`.

This function is used to loop over the Valet results. The function returns true if a result could be returned, false otherwise.

Each result consists of players and outputs. The players are now given as positions, not as absolute positions, so `decl1, decl2, def1, def2` rather than `north, east, south, west`. This is because the output is given for the declaring and defending side separately. (For a passed hand, the assignment is arbitrary.) The fields are in the form of tags as above.

The output has eight fields that contains the components of the Valet score.

- `bool declFlag[2]`: Element 0 is `true` if `decl1` is the declarer, `false` otherwise. Similarly for element 1 and `decl2`. Only one of them will be set; if the hand was passed out, none of them is set.
- `bool defFlag`: `true` if the hand was not passed out, `false` if it was.
- `bool leadFlag[2]`: Element 0 is `true` if `def1` made an opening lead, `false` otherwise. Similarly for element 1 and `def2`. Only one of them will be set; if no lead was made, none of them is set. This can happen either if the hand was passed out, or if no lead was provided.
- `float overallDecl`: The overall score for the declaring side, analogous to the Butler score.

- `float bidScoreDecl`: The bidding score for the declaring side. The bidding score for the defending side is the negative of this, so you can calculate that for yourself.
- `float playScoreDecl[2]`: The play scores for each of the declarers. The value is zero for a player who was not declarer.
- `float leadScoreDef[2]`: The lead scores for each of the defenders. The value is zero for a player who was not on lead (or for whom a lead was not registered). The value is not calculated if the control `leadFlag` is set to `false`.
- `float restScoreDef[2]`: The remaining defensive score. If no lead was given, then this is the entire defensive score which is the negative of the play score for the declaring side. Otherwise it is the difference between the overall defensive score (the negative of the play score) and the lead score.

2.11 VALETGETNEXTSCOREBYNUMBER

Arguments: `struct PositionsNumberType * players,`
`struct OutputResultType * output.`
Return value: `bool`.

This function is also used to loop over the Valet results. The function returns true if a result could be returned, false otherwise.

The function is entirely analogous to `ValetGetNextScoreByTag()`, but the players are given as unsigned integers and not as tags.

2.12 VALETERRORMESSAGE

Arguments: `int code`
`char line[160].`
Return value: `void`.

If you get an error, you can identify it by its number. You can also turn it into a text message with more detail by calling `ValetErrorMessage()`. This may help you to find the location and type of the input error.

The message is returned in the `line` (for which you must provide storage). You can print this string.

3 Return Codes

Value	Meaning
1	Success (RETURN_NO_FAULT)
-1	General error
-2	Not a valid valet mode
-3	Not a valid number of token in an input line
-10	Round number in an input line is not in range
-11	Board number is not in range
-12	Board number changed within a single board
-13	North is not a known or valid player
-14	East is not a known or valid player
-15	South is not a known or valid player
-16	West is not a known or valid player
-17	Contract could not be parsed
-18	Number of tricks taken is not in range
-19	Contract lead is invalid
-20	Contact level is invalid
-21	Contract denomination is invalid
-22	Contract multiplier is invalid
-23	Declarer is invalid
-24	Lead denomination is invalid
-25	Lead rank is invalid