UPI (Universal Poker Interface) documentation. This protocol is used by PioVIEWER and other tools to communicate with PioSOLVER. The architecture was inspired by UCI (http://en.wikipedia.org/wiki/Universal_Chess_Interface). Similarly to UCI, UPI allows programmers to create their own tools based on PioSOLVER engine as well as easy scripting for non-programmers.

LAST EDITED: August 2th 2016
(PioSOLVER 1.9)

1)NodeID - those are constructed as follows:
-r = root node
-r:0 = first decision node
-r:0:b100:c:8h:c:b50:As - actions are separated by a colon (":"), bets are always a cumulative amount invested by a player so far; c stands for a check or call; b stands for a bet or raise (always to cumulative amount); cards or full flops are written as they appear in the tree

2)Command arguments:
-are separated by spaces
-if you want to use an argument with a space in it (for example as a file path), use quotation marks, like this: "C:\docs and settings\my file name.cfr"

3)General remarks for programmers working with PioSOLVER:
-In PioViewer: Tools->Configuration->enable logging writes all the Viewer <-> Solver communication to log.txt file
-In pioviewer.settings file, under [Debug] add ConsoleEnabled=true line, like this:
[Debug]
DebugMode=false
ConsoleEnabled=true

After that the viewer starts in console enable mode and you can see the communication as it happens in real time
-When you try to understand how to obtain data or results from the solver the best way is to follow the above mentioned points and do it in PioViewer. PioViewer is not a privileged program in any way, it uses the very same text interface which is available to you.

Request	Arguments	Response format	Description	Possible errors
GENERAL				
is_ready	n/a	is_ready ok!	useful for checking if solver is initialized	
go	[n seconds | steps]	go ok! Solver will emit: SOLVER: started once running	optional argument runs solver for n seconds or steps; if omitted solver will run indefinitely (unless accuracy is set and reached)	
stop	n/a	stop ok! Solver will emit: SOLVER: stopped once stopped	stop signal and waits for solver to stop (SOLVER: stopped) and then issues stop ok!	
show_hand_order	n/a	List poker hands. E.g. “AcAd AdAh AdAs ...”	This command always returns all 1326 hands in the same order. Other commands return 1326 numbers representing e.g. frequency of hands in a certain spot in this order. Format: 1326 hands separated by “ “: “2d2c 2h2c 2h2d …”	
show_preflop_order	n/a	List of 169 preflop categories	As used for calc_eq_preflop	
solver_time	n/a	a float	The time since the current solver instance started in seconds	
exit	n/a	n/a	kills solver’s process	
show_settings	n/a	name: value (multiline)	accuracy: (solver stops after reaching it) thread_no: how many threads are allowed (0	

			is as much as possible) info_freq: how often sovler outputs results of the computation; default 25 (which means 25 steps for each player or 50 overall) step: starting multiplier for step size of the algorithm hopeless_thres: when the algorithm starts adjusting (if last improvement is less than hopeless_thres then adjust) adjust_mode: 0 = manual, 1 = autoadjust	
show_tree_params	n/a	name: value (multiline)	board: cards (in readable format) or “not set” pot: 3 integers representing starting pot bet_sizes: bet sizes used for automatically building a tree; absolute values (not % of the pot) donk_bet: TRUE/FALSE if flop donkbet should be included in the tree (omitting it makes the tree almost 2x smaller)	
load_script	filename	load_script ok!	reads commands from file; line after line and executes them as if they were inserted on stdin; after reaching EOF goes back to	

			receiving input from stdin	
bench	n/a	Time taken: float	builds a tree about 2.7GB big, runs 6 full iterations on it; returns running time for those (but not time taken to build/free the tree). Designed to take about 30 seconds on i7 quad	
set_threads	int	set_threads ok!	sets number of threads used by solver and all functions requaring tree traversal (like show_range, calc_ev etc.)	
set_info_freq	int	set_info_freq ok!	sets how often solver info is released (defeault is every 25 steps)	
set_recalc_accuracy	float float float	set_recalc_accuracy ok!	sets accuracy for flop/turn/river recalculations (which occur when browsing incomplete tree or calling solve_partial)	
set_adjust_strat	float float int	set_adjust_strat ok!	1st argument: starting step 2nd argument: hopeless_thres 3rd argument: adjust_mode (0: manual, 1: auto)	
set_isomorphism	int int	set_isomorphism ok!	1st argument - isomorhism on/off for flop trees 2nd argument - isomorphism on/off for turn trees the default is on on the flop off on the turn	
take_a_break	int	take_a_break ok!	stops the solver (if it's running) and waits n seconds before going back to reading the input	
is_tree_present	n/a	true | false	returns true if a tree the solver is operating	

			on exists	
----------	----------	----------	----------	----------
BUILDING/DELETING TREES				
set_range	OOP | IP 1326floats	set_range ok!	sets range for IP/OOP in a global state	
set_eff_stack	int	Set_eff_stack ok!	Sets effective stack. This is used in tree building to recognize which nodes are all-in nodes and which aren't.	
set_board	As Kh 7d etc.	set_board ok!	sets board in a global state	
set_bet_sizes (deprecated)	b1 … bn integers	set_bet_sizes ok!	sets bet sizes used for constructing a tree	
set_pot	int int int	set_pot ok!	sets starting pot	
set_donk_bet (deprecated)	0 | 1	set_donk_bet ok!	if flop donk_bet should be included when constructing a tree	
set_accuracy	float [chips | fraction]	set_accuracy ok!	accuracy at which solver stops; default is 0 (never). If optional argument is provided the value is treated either as absolute value (chips) or fraction of the pot (fraction). "chips" is the default.	
build_tree	n/a	build_tree ok!	build a tree based on a config created by calling add_line and remove_line	
build_tree_old (deprecated)	n/a	build_tree ok!	uses board/ranges/bet_sizes/pot/donk_bet (in the future maybe more parameters) to	

			build a tree; all those have to be set before calling build_tree	
rebuild_tree (deprecated)	n/a	rebuild_tree ok!	uses tree structure of current tree; refills boards (using current state.board at root) and resets strategies	
load_tree	filename	load_tree ok!	Loads the tree from the content of the save file into memory. State.root points to this tree	i/o errors, file format error, out of memory
dump_tree	Filename [full | no_turns | no_rivers |]	dump_tree ok!	saves current tree to disc; if optional argument is provided (no_turns or no_rivers) a small save will be made while the whole tree is preserved in memory.	
free_tree	n/a	free_tree ok!	deletes current tree and frees the memory (as of now still has slight memory leak about 300kb per tree)	
add_line	series of numbers representing bet sizes (cumulative)	add_line ok!	0 30 30 30 90 represents a check OOP, bet IP, a call, a check on the turn and a bet of 60 (90 total invested). See more examples in sample scripts.	
remove_line	series of numbers representing a line to remove	remove_line ok!	remove_line uses the same syntax as add_line. It's possible to remove calls and folds as well: 0 30 0 removes a fold for OOP player in response to c-bet	
cut_line	nodeID	cut_line ok!	removes a line from the tree (that is all the branches for all the runouts leading to specified line)	

force_line	series of numbers representing bet sizes	force_line ok!	removes all the lines which don't lead to the one being forced (so any line of which forced line isn't a prefix of)	
---	---	---	---	---
PREFLOP TREE BUILDING				
build_preflop_tree	n/a	Build_preflop_tree ok!	Builds a pure preflop tree using a descriptioin as defined by add_preflop_line's commands.	
add_preflop_line	Series of numbers representing bet sizes (cumulative)	Add_preflop_line ok!	Identical as add_line but builds a pure preflop tree (one street)	
clear_preflop_lines	n/a	Clear_preflop_lines ok!	Clears preflop tree structure created by add_preflop_line	
remove_preflop_line	Series of numbers representing a line to remove	Remove_preflop_line ok!	Same as remove_line but for preflop	
add_to_subset	Float card card cards	Add_to_subset ok!	Adds a specified board with specified weight to a flop subset; that subset can be then used to build a full preflop tree with selected flops	
reset_subset	n/a	Reset_subset ok!	Clears the current flop subset	
show_subset	n/a	(multiline) weight1 board1 weight2 board2 ….	Prints current flops with weights from the flop subset.	

add_schematic_tree	nodeID	Add_schematic_tree ok!	Attaches a postflop abstraction to a chosen preflop exit. NodeID must point to a SPLIT_NODE which is a valid preflop exit. Current schematic tree (one created by add_line commands) is attached. The way to use this command is to build a postflop tree with add_line commands, attach it to a chosen preflop exit; reset it, build another postflop tree, attach it to another preflop exit etc.	
add_all_flops	n/a	Add_all_flops ok!	Uses flops in a current flop subset to build a full preflop tree.	
TRAVERSING THE TREE				
show_node	nodeID	(multiline) nodeID NODE_TYPE board pot children_no flags: f1 … f2	the number of flags can vary from 0 to 64; they are separated by space	
show_children	nodeID	(multiline) child 0: …. ….. child 1: ….	… are in the same format as show_node	

		….		
show_range	OOP | IP [nodeID]	1326 floats	range in given node; dead hands have weight 0 (last part doesn't work yet) If only one argument is given then IP/OOP range from solver state is shown (the one set by set_range)	
show_strategy	nodeID	(multiline) 1326 floats 1326 floats ….	n line represents frequency of n'th action with i'th hand Error if nodeID doesn't represent decision node	
show_strategy_pp	nodeID	human readable sorted (by equity vs ALL) output	Error if nodeID doesn't represent decision node	
calc_ev	OOP | IP nodeID	(2 lines) 1326 floats 1326 floats	EV in the first one, matchups in 2nd	
calc_ev_line	OOP | IP nodeID	(2lines) 1326 floats	EV of a line, that is combined EV of the line on all the runouts following it; the format is the same as in calc_ev	
calc_ev_pp	OOP | IP nodeID	wins/matchups in human readable format		

calc_results	n/a	(multiline) running time: float EV OOP: float EV IP: float OOP's MES: float IP's MES: float exploitable for: float	calculates EV's/MES'es in root and prints the whole info;	
calc_global_freq	nodeID	a float	Returns probability of reaching a given node	
calc_line_freq	nodeID	a float	Returns probability of reaching a given line (similar to calc_ev_line but returns one number for probability and no evs)	
---	---	---	---	---
OTHER FEATUERS				
calc_eq	OOP | IP	(2lines) 1326 floats 1326 floats	eq in the first line, matchups in 2nd board/ranges taken from solver state (set_range, set_board to set)	
calc_eq_pp	OOP | IP	in human readable form		
calc_eq_node	OOP | IP nodeID	(3lines) 1326 floats 1326 floats 1 float	calculate equity for given player assuming ranges in given node; returned values: eqs/matchups/overall	
calc_eq_preflop	OOP | IP	(3lines) 169 floats 169 floats 1 float	Calculates preflop equity from IP/OOP range as set by set_range command. It assumes weights for isomorphic (7s6s = 7h6h) combos is the same. The results are	

			unpredictable/incorrect if that's not the case. Output format: 1line: equities 2line: matchups 3line: total	
show_all_freqs	global | local [pp]	(multiline) All lines with corresponsing frequencies	Lists all lines in the tree with corresponding frequncies either local (probability of taking last action) or global (probability of this line being played); optional argument pp (pretty print) makes the output easier to read for a human	
stdoutredi	filename	-	redirects stdout to file	
stdoutredi_append	filename	-	redirects stdout to file but appends it instead of overwriting	
stdoutback	n/a	stdoutback ok!	standard output back to console (might be useful when using text interface, not for GUI)	
estimate_tree deprecated	n/a	uint64	estimates size of the full tree before ranges are initizalized	
estimate_schematic_tree	n/a	uint64	Estimates size of a tree based on addline/remove interface; the results is in Megabytes	
wait_for_solver	n/a	waits until solver stops before reading rest of the commands from stdin	useful for scripts if one wants to solve multiple trees (or solve and save for example);	

show_memory	n/a	(2 lines) uint64 uint64	shows total physical memory and totla available physical memory (at the moment)	
needed_memory	n/a	(2lines) uint64 uint64	first row: how much physical memory there is; second row: how much is needed to build a tree with current parameters	
ignore_mem_check	on | off	ignore_mem_check ok!	if turned on the check for available RAM is not performed. It may result in a computer slowing down to a crawl if Windows starts using a swap file.	
skip_if_done	filename label	skip_if_done ok!	checks if a filename exists if yes, skips all the lines in the script until label is encountered	
LABEL:	labelname	none	labels a place in the script; this is used by skip_if_done command above (remember to include a space after a colon and before the label name)	
---	---	---	---	---
TREE OPERATIONS				
forget	turns | rivers	forget ok!	fills cache and forgets rivers or turns+rivers; the tree is usable for browsing but not for solving after that operation	
clear_cache	n/a	clear_cache ok!	clears cached results in the tree;	

rebuild_forgotten_streets (rebuild_all)	n/a	rebuild_all ok!	rebuilds all turns/rivers which were forgotten by forget command; it's spossible to resume solving after that	
solve_partial	nodeID	solver_partial ok!	solve a subtree from given node to accuracy in settings.accuracy	
explo_partial	nodeID	float	returns exploitability for a subtree starting from given node	
lock_node	nodeID	lock_node ok!	locks strategy in given decision node; sets LOCKED flag and unsets COMBO_LOCKED flag	
unlock_node	nodeID	unlock_node ok!	unlocks strategy in given decision node; unsets both LOCKED and COMBO_LOCKED flags	
combo_lock_node	nodeID 1326 ints	combo_lock_node ok!	locks specific combos at given node (given by nodeID). The function expects 1326 0 or 1 ints; 1 = this combo is locked, 0 = this combo is not locked. COMBO_LOCKED flag is set and LOCKED flag is unset.	
set_strategy	nodeID childnum 1326 floats	set_strategy ok!	sets strategy in given decision node for indicated child to provided values;	
normalize_tree	n/a	normalize_tree ok!	normalizes the strategies; there is usually no need to call it as the solver normalizes the tree itself when needed	

set_mes	IP | OOP	set_mes ok!	sets strategy for IP or OOP to most exploitive one in the whole tree	
solve_all_splits	turns | rivers	solver_all_splits ok!	solve all forgotten parts of the tree to accuracy defined in settings.recalc_accuracy (it may take a long time)	
node_count	n/a	multiline string	returns number of nodes of various types on flop/turn/river in human readable format	
show_effective_stack	n/a	int	shows effective stacks in the tree	
round_up_to	IP | OOP chunks_no flop | turn | river	round_up_to ok!	rounds the strategies for given player using specified number of chunks to divide the strategy (4 chunks is 0 0.25 0.5 0.75 1); from the flop up to specified street	
---	---	---	---	---
RANGES & EVALS				
show_category_names	n/a	2lines: hand strength categories draw categories	shows names of hand/draws categories which are then used for range analysis	
show_categories	board	2lines: 1326 ints 1326 ints	it returns categories ids (an order returned by show_category_names); first line is for hand categories like top_pair, 2nd one is for draws	
