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: OCTOBER 11th 2015
(PioSOLVER 1.6 version)

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_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	
build_pf_tree	n/a	build_pf_tree ok!	uses board/ranges/bet_sizes/pot/donk_bet the same way as build_tree_old did; donk_bet = if a call (completing of the blind) ends the betting (donk_bet == 0 -> end the beting, donk_bet == 1 -> don't)	
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	dump_tree ok!	saves current tree to disc	
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_line ok!	remove_line uses the same syntax as add_line. It's possible to remove calls and	

	remove		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)	
---	---	---	---	---
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;	

| ---------- | ---------- | ---------- | ---------- | ---------- |
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	
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	
unluck_node	nodeID	unlock_node ok!	unlocks strategy in given decision node	
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 in a node.	

			LOCKED flag overrides this functionality so make sure not to lock a whole node when using it	
combo_unlock_node	nodeID	combo_unlock_node ok!	Unlocks all the combos in a node and removes COMBO_LOCKED flag.	
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	