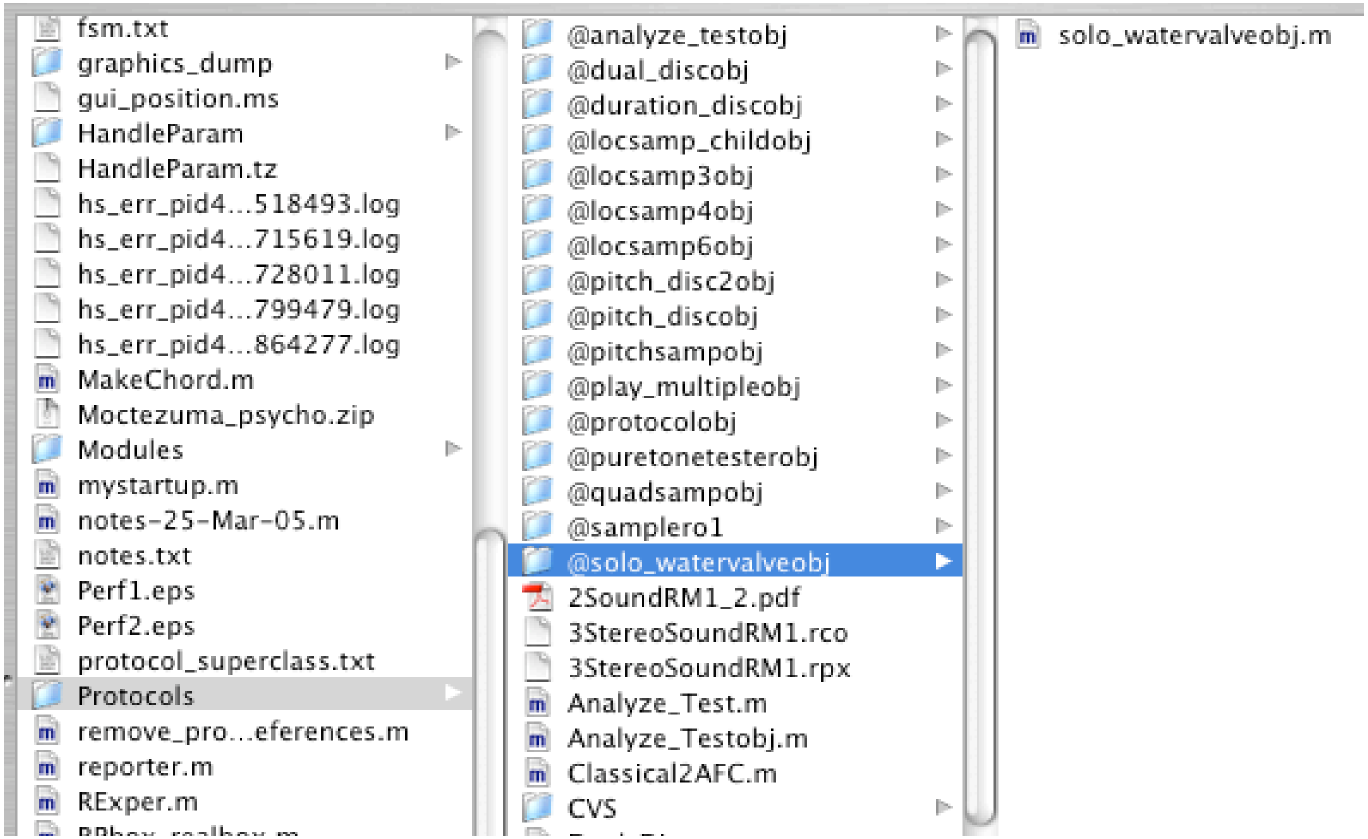
Protocol-writing with the Solo system:

The Basics:
The Water Valve Calibrator

Getting your Bearings

(or, "If only all pathways were this easy")



fsm.txt
graphics_dump ▶
gui_position.ms
HandleParam ▶
HandleParam.tz
hs_err_pid4...518493.log
hs_err_pid4...715619.log
hs_err_pid4...728011.log
hs_err_pid4...799479.log
hs_err_pid4...864277.log
MakeChord.m
Moctezuma_psycho.zip
Modules ▶
mystartup.m
notes-25-Mar-05.m
notes.txt
Perf1.eps
Perf2.eps
protocol_superclass.txt
Protocols ▶
remove_pro...eferences.m
reporter.m
RExper.m

@analyze_testobj ▶
@dual_discobj ▶
@duration_discobj ▶
@locsamp_childobj ▶
@locsamp3obj ▶
@locsamp4obj ▶
@locsamp6obj ▶
@pitch_disc2obj ▶
@pitch_discobj ▶
@pitchsampobj ▶
@play_multipleobj ▶
@protocolobj ▶
@puretonetesterobj ▶
@quadsampobj ▶
@samplero1 ▶
@solo_watervalveobj ▶
2SoundRM1_2.pdf
3StereoSoundRM1.rco
3StereoSoundRM1.rpx
Analyze_Test.m
Analyze_Testobj.m
Classical2AFC.m
CVS ▶

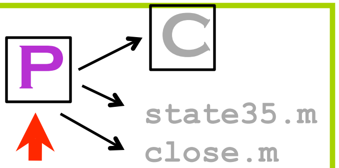
solo_watervalveobj.m

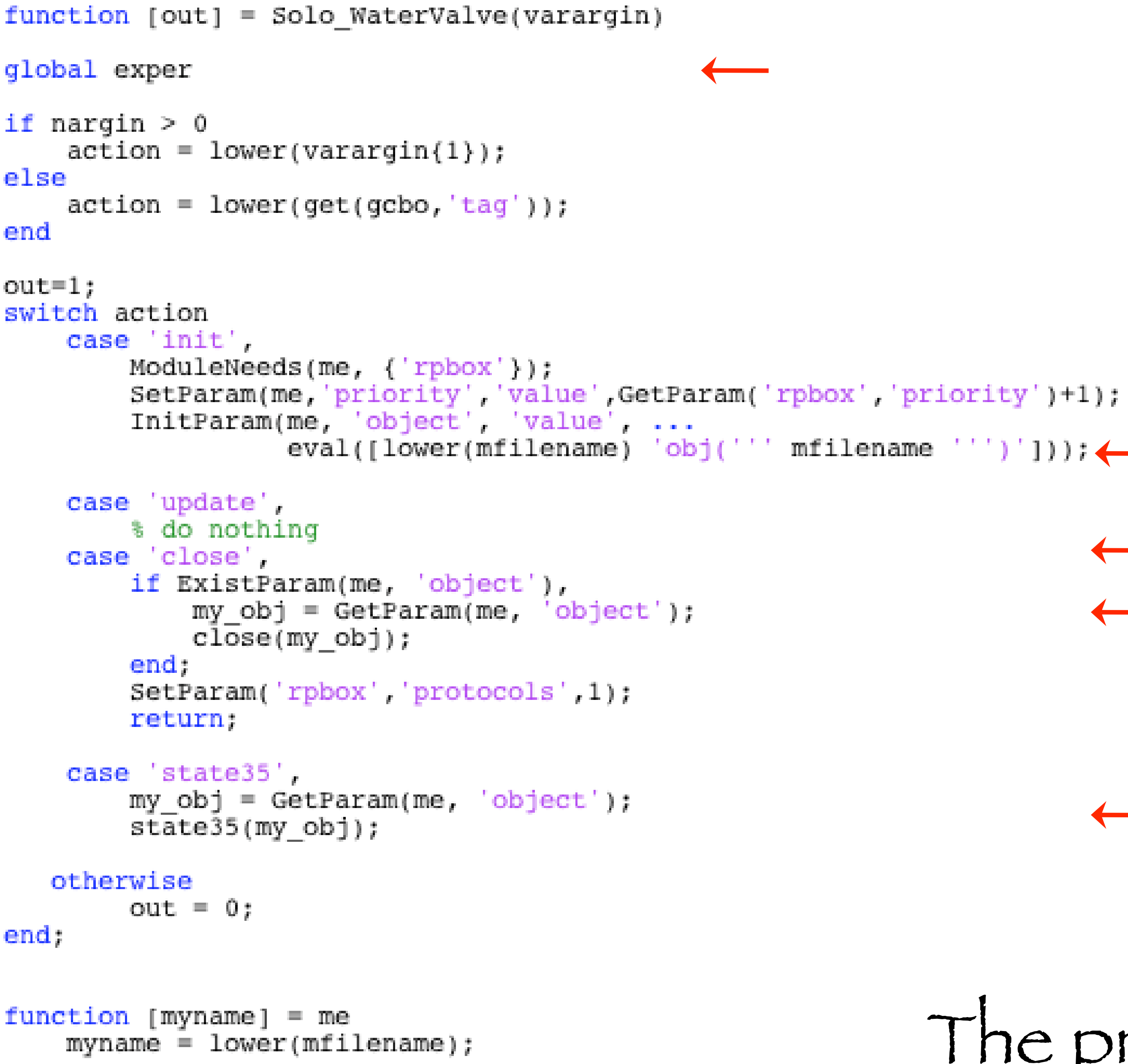
Path to protocol executable: `(root-dir)/ExperPort/Protocols/`

Protocol object: *`protocolname`***`obj.m`**

Object files in: `(1)/@protocolname`**`obj.m`**

solo_watervalveobj.m





```
function [out] = Solo_WaterValve(varargin)

global exper

if nargin > 0
    action = lower(varargin{1});
else
    action = lower(get(gcbo,'tag'));
end

out=1;
switch action
    case 'init',
        ModuleNeeds(me, {'rpbox'});
        SetParam(me,'priority','value',GetParam('rpbox','priority')+1);
        InitParam(me, 'object', 'value', ...
                  eval([lower(mfilename) 'obj(''' mfilename ''')']));

    case 'update',
        % do nothing
    case 'close',
        if ExistParam(me, 'object'),
            my_obj = GetParam(me, 'object');
            close(my_obj);
        end;
        SetParam('rpbox','protocols',1);
        return;

    case 'state35',
        my_obj = GetParam(me, 'object');
        state35(my_obj);

   otherwise
        out = 0;
end;


function [myname] = me
    myname = lower(mfilename);
```

Call to constructor

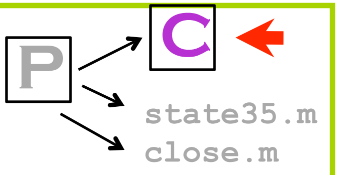
(update files)

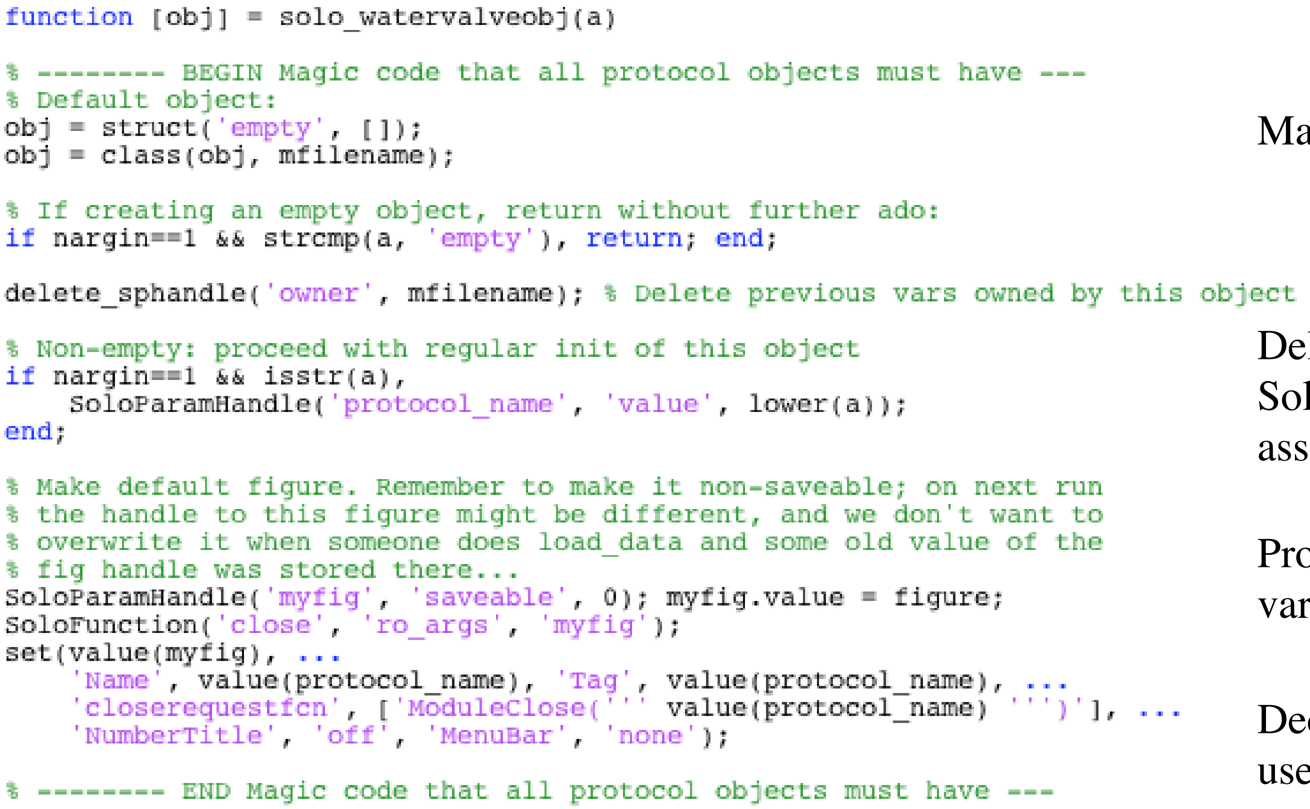
close.m

state35.m

The protocol file

Constructor





```
function [obj] = solo_watervalveobj(a)

% -------- BEGIN Magic code that all protocol objects must have ---
% Default object:
obj = struct('empty', []);
obj = class(obj, mfilename);

% If creating an empty object, return without further ado:
if nargin==1 && strcmp(a, 'empty'), return; end;

delete_sphandle('owner', mfilename); % Delete previous vars owned by this object

% Non-empty: proceed with regular init of this object
if nargin==1 && isstr(a),
    SoloParamHandle('protocol_name', 'value', lower(a));
end;

% Make default figure. Remember to make it non-saveable; on next run
% the handle to this figure might be different, and we don't want to
% overwrite it when someone does load_data and some old value of the
% fig handle was stored there...
SoloParamHandle('myfig', 'saveable', 0); myfig.value = figure;
SoloFunction('close', 'ro_args', 'myfig');
set(value(myfig), ...
    'Name', value(protocol_name), 'Tag', value(protocol_name), ...
    'closerequestfcn', ['ModuleClose(''' value(protocol_name) ''')'], ...
    'NumberTitle', 'off', 'MenuBar', 'none');

% -------- END Magic code that all protocol objects must have ---
```

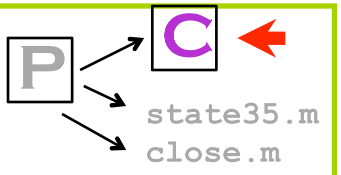
Matlab object creation code

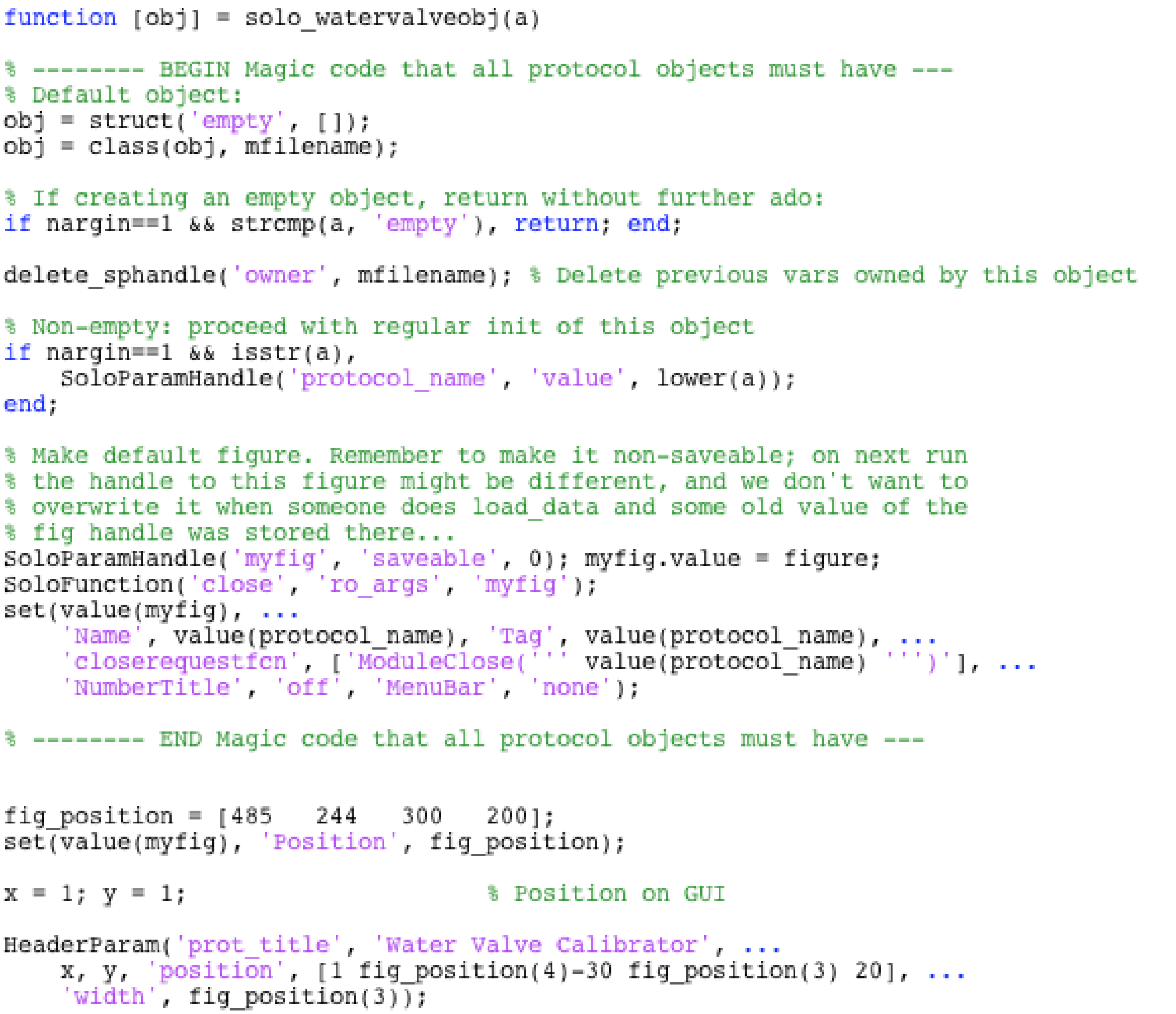
Delete previous SoloParamHandles (SPH) associated with the object

Protocol-nonspecific variables

Declaring a function that uses SPHs

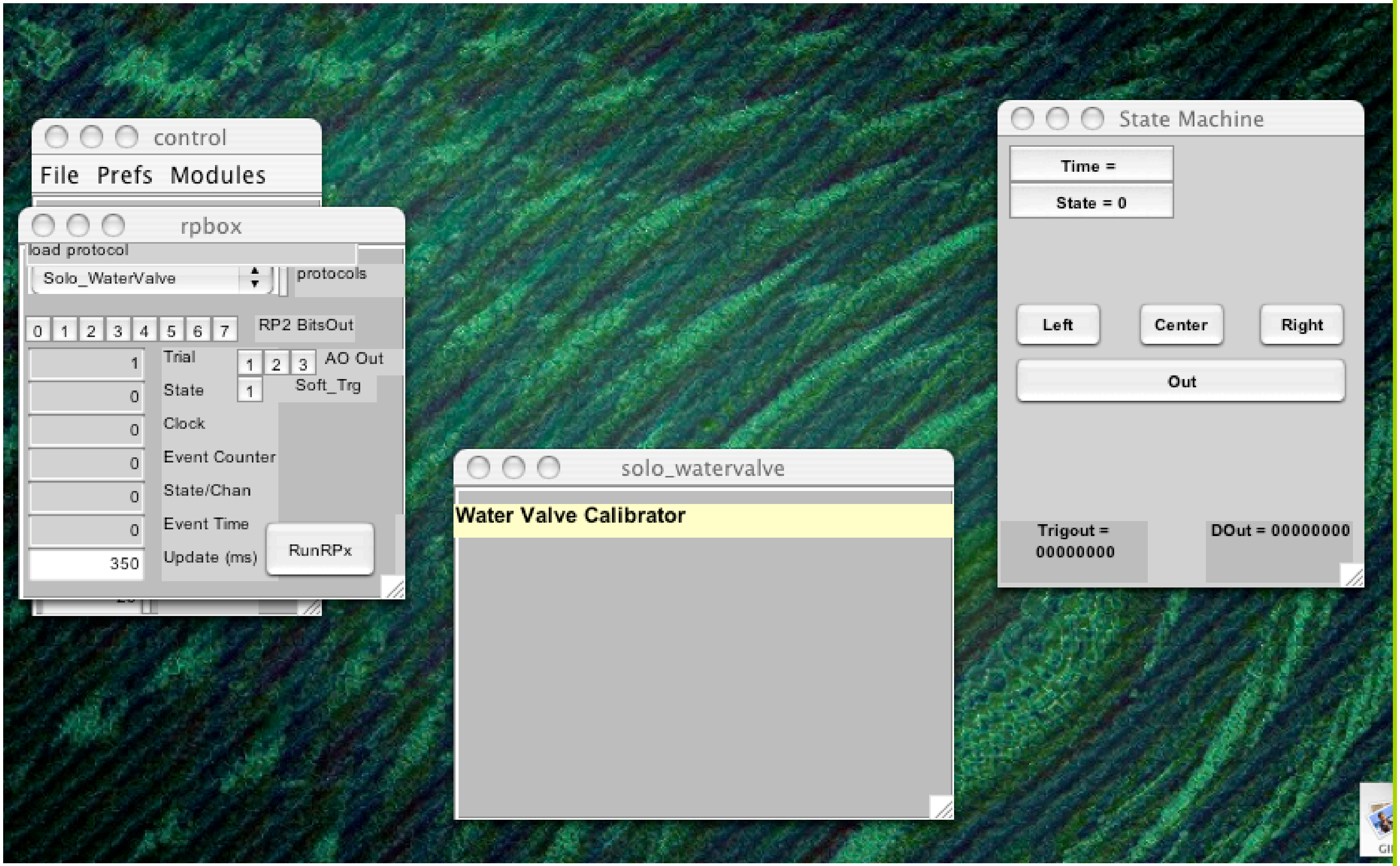
Constructor: Adding UI elements





```
function [obj] = solo_watervalveobj(a)

% -------- BEGIN Magic code that all protocol objects must have ---
% Default object:
obj = struct('empty', []);
obj = class(obj, mfilename);

% If creating an empty object, return without further ado:
if nargin==1 && strcmp(a, 'empty'), return; end;

delete_sphandle('owner', mfilename); % Delete previous vars owned by this object

% Non-empty: proceed with regular init of this object
if nargin==1 && isstr(a),
    SoloParamHandle('protocol_name', 'value', lower(a));
end;

% Make default figure. Remember to make it non-saveable; on next run
% the handle to this figure might be different, and we don't want to
% overwrite it when someone does load_data and some old value of the
% fig handle was stored there...
SoloParamHandle('myfig', 'saveable', 0); myfig.value = figure;
SoloFunction('close', 'ro_args', 'myfig');
set(value(myfig), ...
    'Name', value(protocol_name), 'Tag', value(protocol_name), ...
    'closerequestfcn', ['ModuleClose(''' value(protocol_name) ''')'], ...
    'NumberTitle', 'off', 'MenuBar', 'none');

% -------- END Magic code that all protocol objects must have ---


fig_position = [485   244   300   200];
set(value(myfig), 'Position', fig_position);

x = 1; y = 1;                          % Position on GUI

HeaderParam('prot_title', 'Water Valve Calibrator', ...
    x, y, 'position', [1 fig_position(4)-30 fig_position(3) 20], ...
    'width', fig_position(3));
```

Previewing Greatness to Come ...



load protocol: Solo_WaterValve

Soft_Trg: 1

1	Trial
0	State
0	Clock
0	Event Counter
0	State/Chan
0	Event Time
350	Update (ms)

Water Valve Calibrator

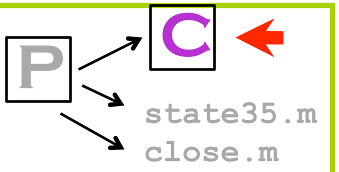
Time =

State = 0

Trigout = 00000000

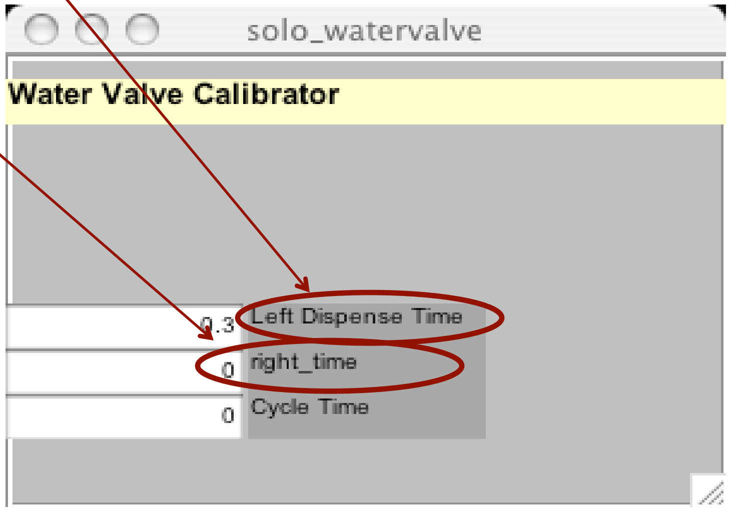
DOut = 00000000

Constructor: Adding UI elements (2)



state35.m
close.m

```
fig_position = [485   244   300   200];
set(value(myfig), 'Position', fig_position);

x = 1; y = 1;                      % Position on GUI

next_row(y, 1.5);
NumeditParam('cycle_time', 0, x, y, 'label', 'Cycle Time');next_row(y);
NumeditParam('right_time', 0, x, y);next_row(y);
NumeditParam('left_time', 0.3, x, y, 'label', 'Left Dispense Time', ...
    'TooltipString', 'Time (in seconds) to open the left water valve per cycle');
next_row(y);



HeaderParam('prot_title', 'Water Valve Calibrator', ...
    x, y, 'position', [1 fig_position(4)-30 fig_position(3) 20], ...
```

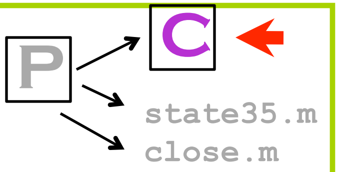

solo_watervalve

Water Valve Calibrator

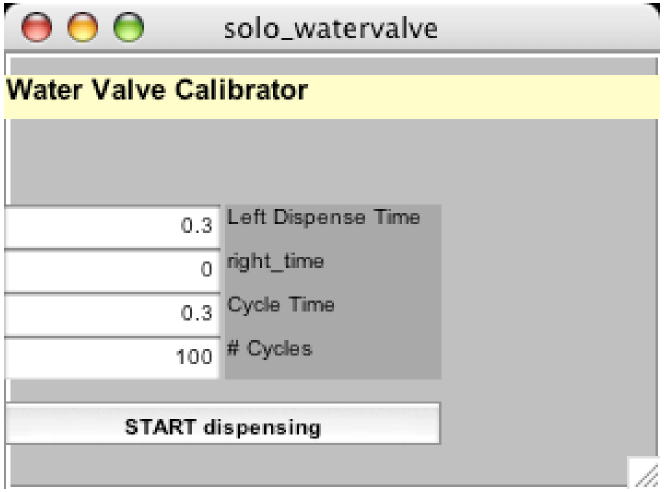
0.3 Left Dispense Time
0 right_time
0 Cycle Time

- Optional UI features (labels, tooltips)
- Positioning

Constructor: Add Dispense Control



```
next_row(y);
ToggleParam('go', 0, x, y, ...
     'OnString', 'STOP dispensing', ...
     'OffString', 'START dispensing');

next_row(y, 1.5);
NumEditParam('num_cycles', 100, x, y, 'label', '# Cycles'); next_row(y);
NumeditParam('cycle_time', 0, x, y, 'label', 'Cycle Time');next_row(y);
NumeditParam('right_time', 0, x, y);next_row(y);
NumeditParam('left_time', 0.3, x, y, 'label', 'Left Dispense Time'
```

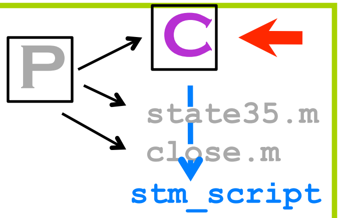

solo_watervalve

Water Valve Calibrator

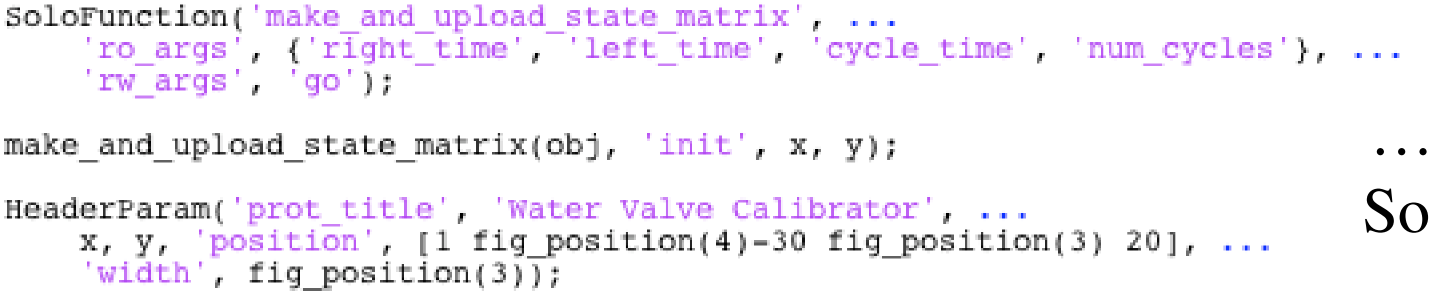
0.3	Left Dispense Time
0	right_time
0.3	Cycle Time
100	# Cycles

Look at *HandleParam/*.m* for available features
e.g. *HandleParam/EditParam.m*

The constructor initialises the state matrix



Declaring …


```
SoloFunction('make_and_upload_state_matrix', ...
    'ro_args', {'right_time', 'left_time', 'cycle_time', 'num_cycles'}, ...
    'rw_args', 'go');

make_and_upload_state_matrix(obj, 'init', x, y);

HeaderParam('prot_title', 'Water Valve Calibrator', ...
    x, y, 'position', [1 fig_position(4)-30 fig_position(3) 20], ...
    'width', fig_position(3));
```

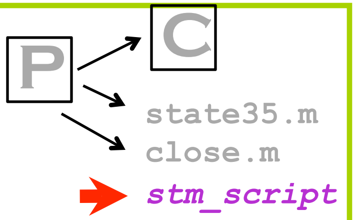
… and calling SoloFunctions

The constructor initialises all peripheral sections of a protocol. *e.g.*

Deciding trial sides (*SidesSection.m*)

Constructing sounds (*ChordSection.m*)

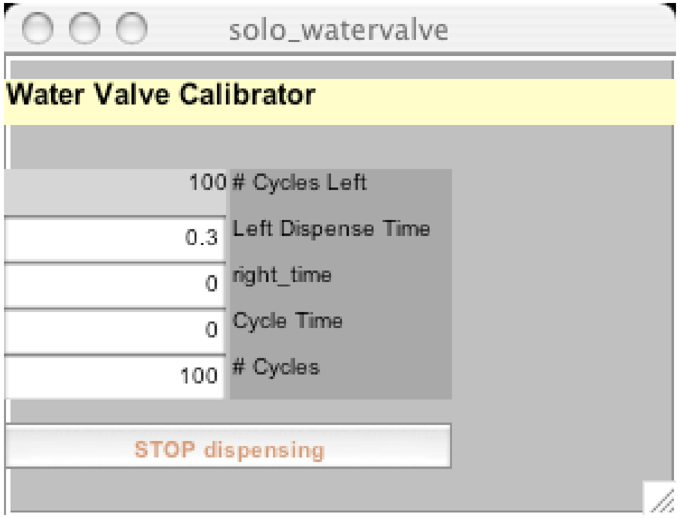
Programming the state matrix: Skeleton for specialised files



```
function [] = make_and_upload_state_matrix(obj, action, x, y)

GetSoloFunctionArgs;

switch action
    case 'init'
    % initialises GUI elements owned by this function
    DispParam('cycles_left', value(num_cycles), x, y, ...
        'label', '# Cycles Left'); next_row(y);


    case 'next_matrix'
    % used in all subsequent calls (note: no initialisation
    % of GUI elements occurs here)

    otherwise
        error('Invalid action!');
end;

% real state matrix definition goes here
```

How a SoloFunction gets registered SPHs **during each call**

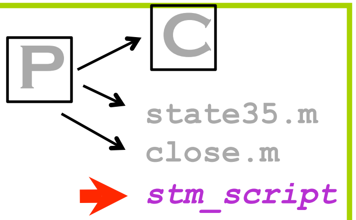
Adding UI elements to the main figure (controlled by this script)

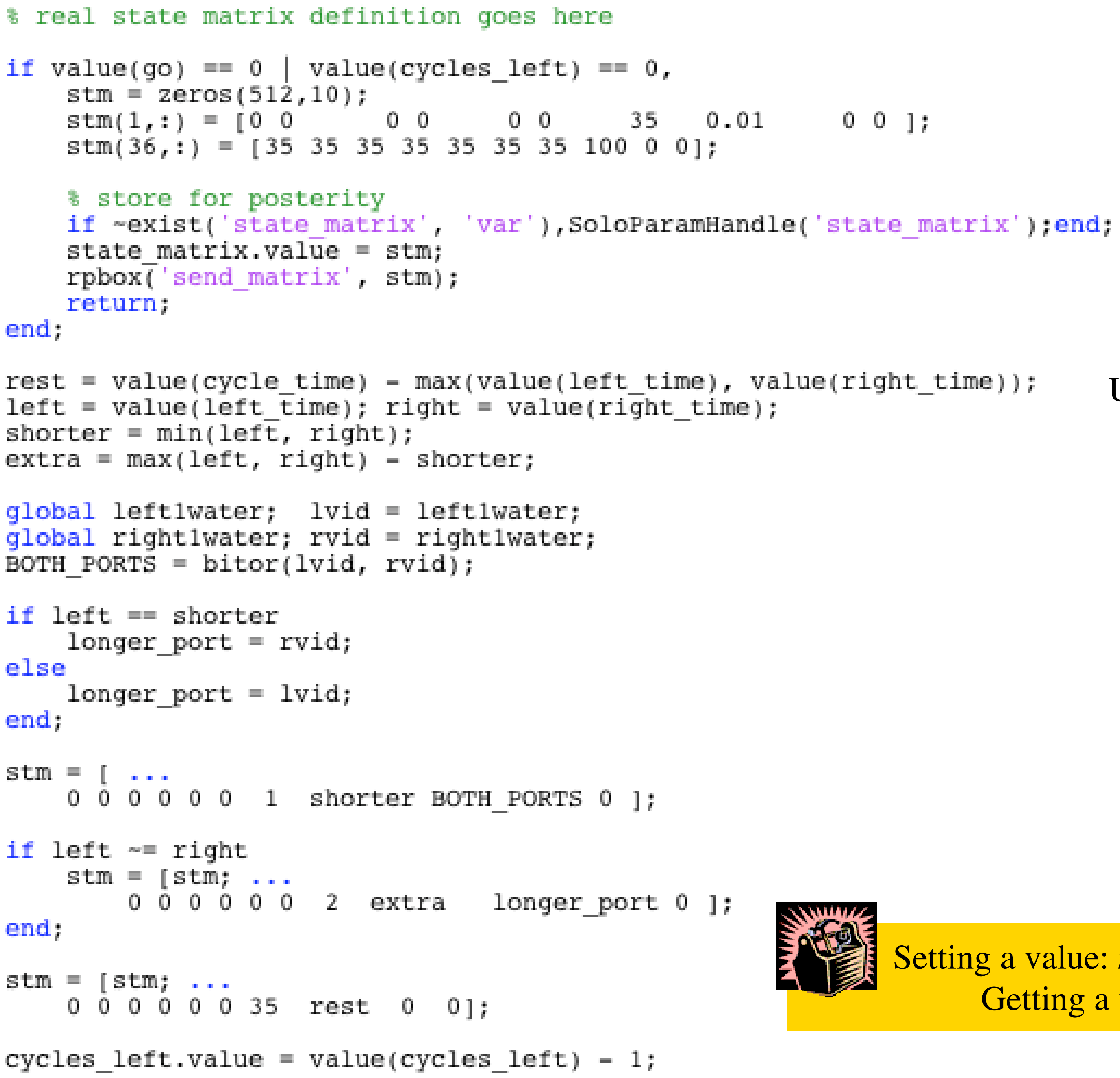


solo_watervalve

Water Valve Calibrator

100 # Cycles Left
0.3 Left Dispense Time
0 right_time
0 Cycle Time
100 # Cycles





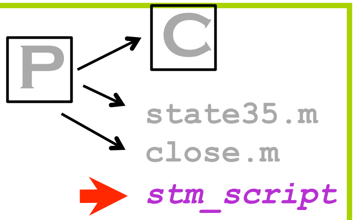
```
% real state matrix definition goes here

if value(go) == 0 | value(cycles_left) == 0,
    stm = zeros(512,10);
    stm(1,:) = [0 0      0 0      0 0      35    0.01      0 0 ];
    stm(36,:) = [35 35 35 35 35 35 35 100 0 0];

    % store for posterity
    if ~exist('state_matrix', 'var'),SoloParamHandle('state_matrix');end;
    state_matrix.value = stm;
    rpbox('send_matrix', stm);
    return;
end;

rest = value(cycle_time) - max(value(left_time), value(right_time));
left = value(left_time); right = value(right_time);
shorter = min(left, right);
extra = max(left, right) - shorter;

global left1water;  lvid = left1water;
global right1water; rvid = right1water;
BOTH_PORTS = bitor(lvid, rvid);

if left == shorter
    longer_port = rvid;
else
    longer_port = lvid;
end;

stm = [ ...
    0 0 0 0 0 0  1  shorter BOTH_PORTS 0 ];

if left ~= right
    stm = [stm; ...
        0 0 0 0 0 0  2  extra   longer_port 0 ];
end;

stm = [stm; ...
    0 0 0 0 0 0 35  rest  0  0];

cycles_left.value = value(cycles_left) - 1;
```

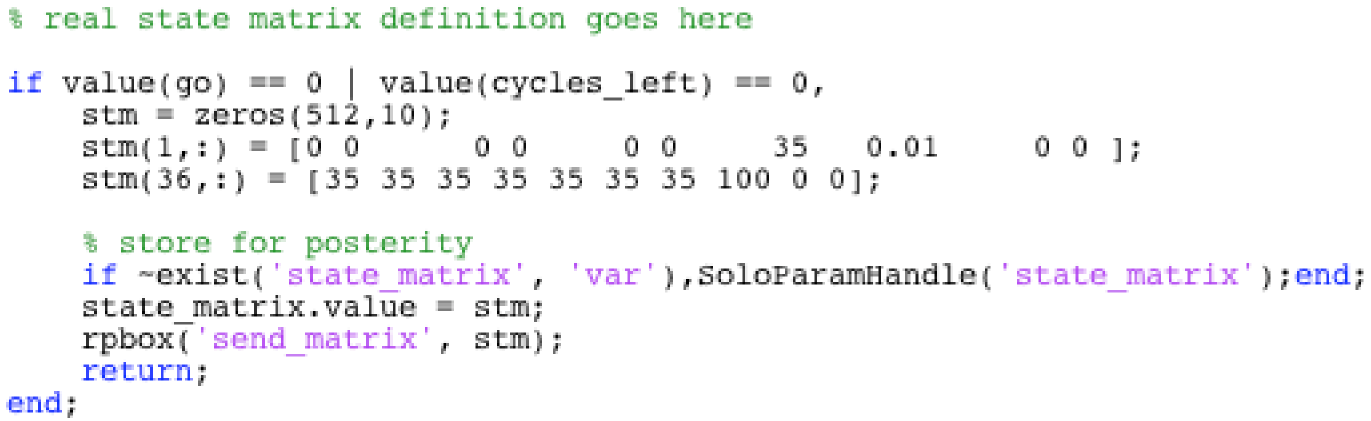
Using read-only args

Using flags set in `mystartup.m`

Setting a value: *myvar.value = 22/7*;
Getting a value: *value(myvar)*

Talking to the RPBox: *send_matrix*



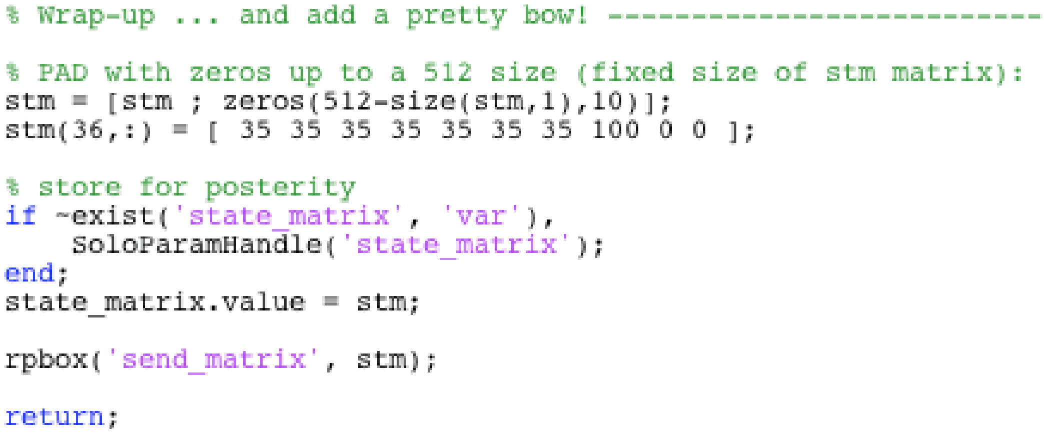


```
% real state matrix definition goes here

if value(go) == 0 | value(cycles_left) == 0,
    stm = zeros(512,10);
    stm(1,:) = [0 0      0 0      0 0      35   0.01      0 0 ];
    stm(36,:) = [35 35 35 35 35 35 35 100 0 0];

    % store for posterity
    if ~exist('state_matrix', 'var'),SoloParamHandle('state_matrix');end;
    state_matrix.value = stm;
    rpbox('send_matrix', stm);
    return;
end;
```

...

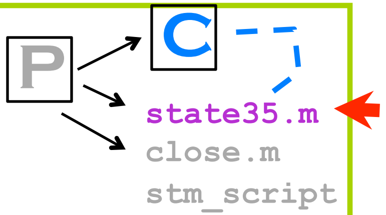
...

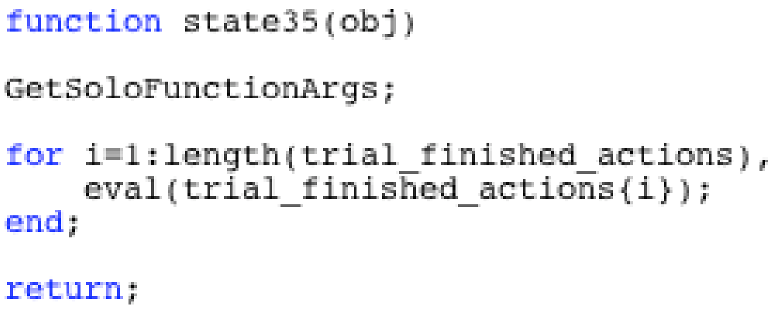
...



```
% Wrap-up ... and add a pretty bow! -------------------------

% PAD with zeros up to a 512 size (fixed size of stm matrix):
stm = [stm ; zeros(512-size(stm,1),10)];
stm(36,:) = [ 35 35 35 35 35 35 35 100 0 0 ];

% store for posterity
if ~exist('state_matrix', 'var'),
    SoloParamHandle('state_matrix');
end;
state_matrix.value = stm;

rpbox('send_matrix', stm);

return;
```

state35.m: Executed at the end of trials

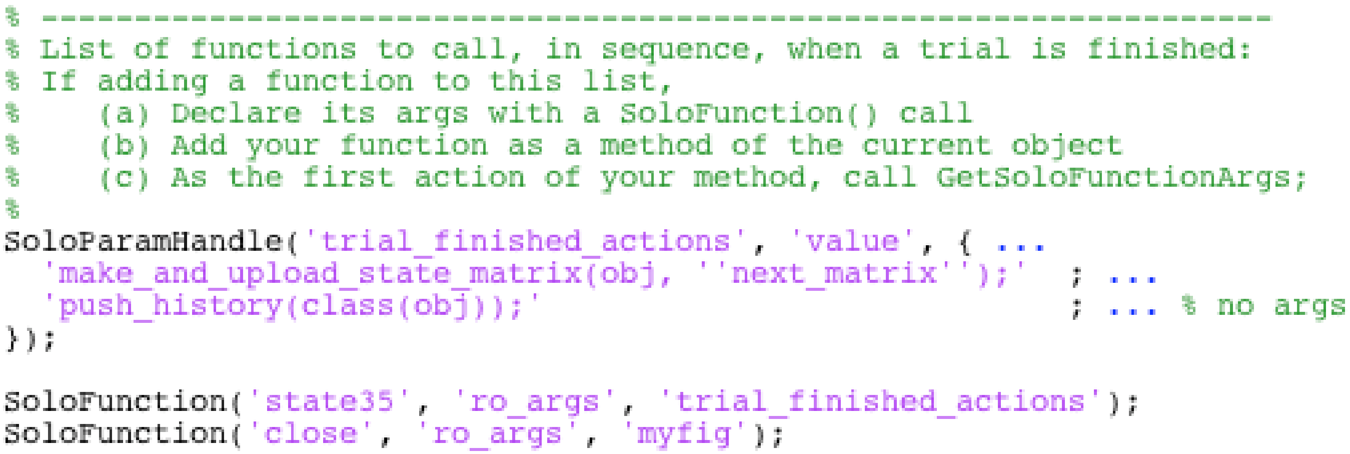




```
function state35(obj)

GetSoloFunctionArgs;

for i=1:length(trial_finished_actions),
    eval(trial_finished_actions{i});
end;

return;
```

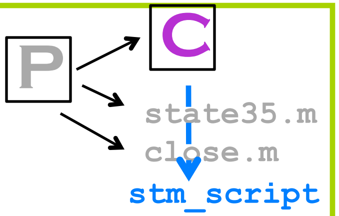
trial_finished_actions is an SPH!

Constructor



```
% ------------------------------------------------------------------
% List of functions to call, in sequence, when a trial is finished:
% If adding a function to this list,
%    (a) Declare its args with a SoloFunction() call
%    (b) Add your function as a method of the current object
%    (c) As the first action of your method, call GetSoloFunctionArgs;
%
SoloParamHandle('trial_finished_actions', 'value', { ...
   'make_and_upload_state_matrix(obj, ''next_matrix'');'  ; ...
   'push_history(class(obj));'                            ; ... % no args
});

SoloFunction('state35', 'ro_args', 'trial_finished_actions');
SoloFunction('close', 'ro_args', 'myfig');
```

Not just a pretty face: Callbacks



P

C

state35.m

close.m

stm_script

```
next_row(y);
ToggleParam('go', 0, x, y, ...
    'OnString', 'STOP dispensing', ...
    'OffString', 'START dispensing');
set_callback(go, {'make_and_upload_state_matrix', 'next_matrix'});

next_row(y, 1.5);
NumEditParam('num_cycles', 100, x, y, 'label', '# Cycles'); next_row(y);
set_callback(num_cycles, {'make_and_upload_state_matrix', 'set_cycles'});
set_callback({left_time, right_time, cycle_time}, ...
    {'make_and_upload_state_matrix', 'check_cycle_time'});
```

```
switch action
    case 'init'
        % initialises GUI elements owned by this function
        DispParam('cycles_left', value(num_cycles), x, y, ...
            'label', '# Cycles Left'); next_row(y);

        make_and_upload_state_matrix(obj,'check_cycle_time');
    case 'next_matrix'
        % used in all subsequent calls (note: no initialisation
        % of GUI elements occurs here)

        if value(cycles_left) == 0
            go.value = 0;
        end;

    case 'set_cycles'
        cycles_left.value = value(num_cycles);
        return;

    case 'check_cycle_time'
        if value(cycle_time) < max(value(left_time), value(right_time))
            cycle_time.value = max(value(left_time), value(right_time));
        end;

        return;

    otherwise
        error('Invalid action!');
end;
```

Not **returning** =>
extra state matrix
generations!

Your rats will

never

be biased

towards one drink port again!