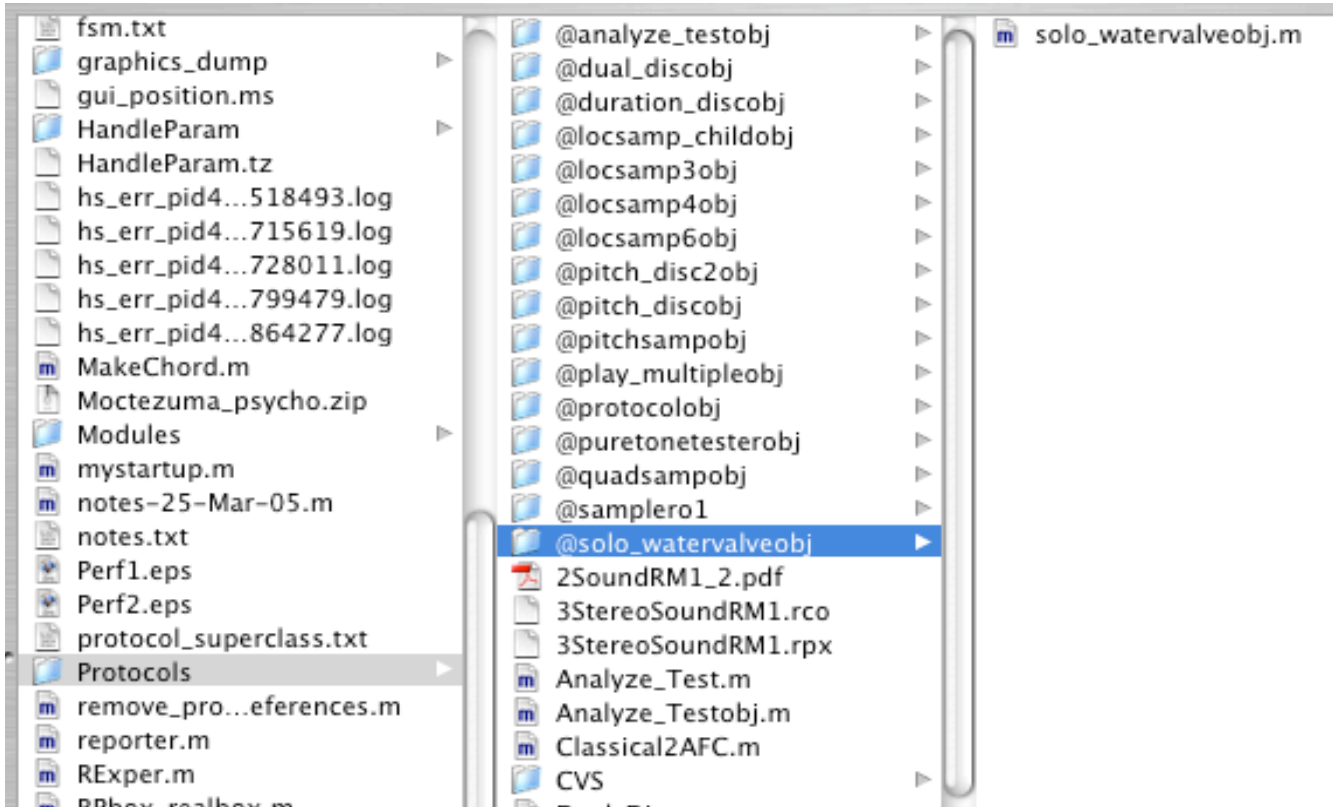
Protocol-writing with the Solo system:

The Basics:
The Water Valve Calibrator

Getting your Bearings

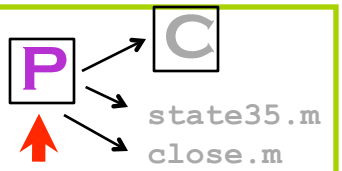


Path to protocol executable: `(root-dir)/ExperPort/Protocols/`

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

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

solo_watervalve.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

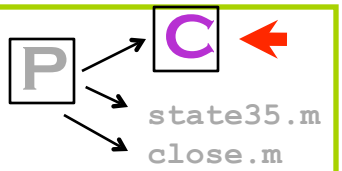
(update files)

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 ---
```

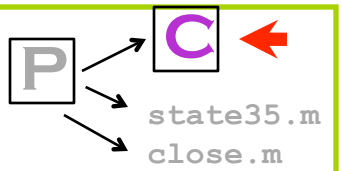
Matlab object creation code

Delete previous SoloParamHandles (SPH) associated with the object

Protocol-nonspecific variables

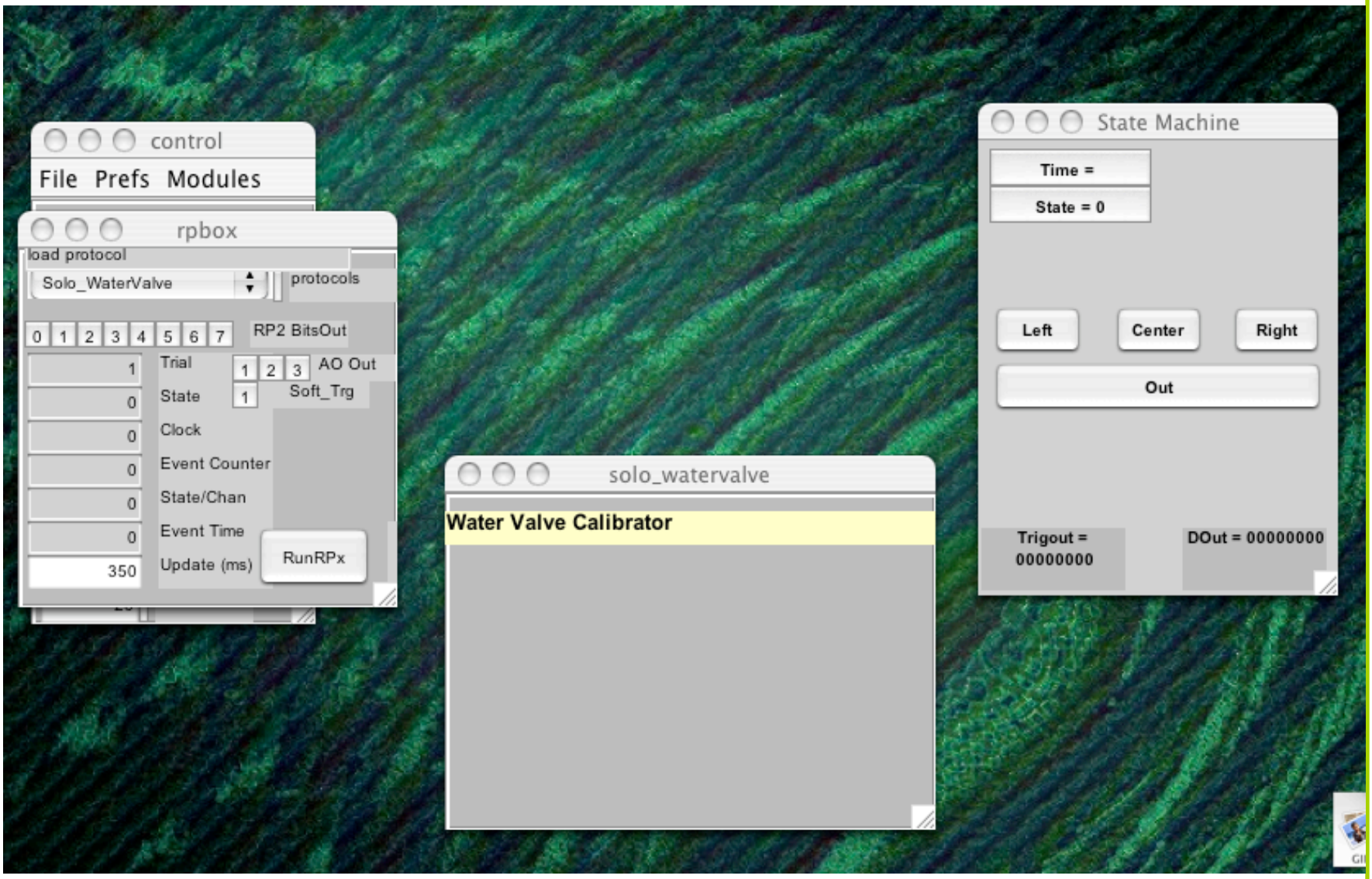
Declaring a function that uses SPHs

Constructor: Adding UI elements

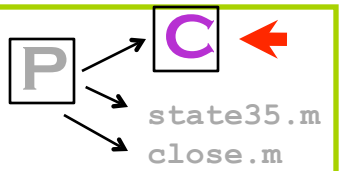



```
function [obj] = solo_watervalveobj(a)

% -------- 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));
```

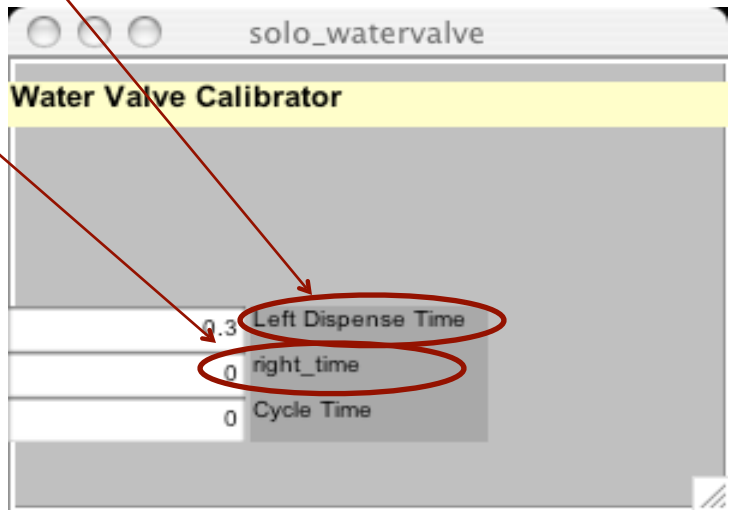
Previewing Greatness to Come …



Constructor: Adding UI elements (2)

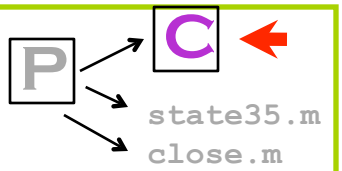



```
fig_position = [485   244   300   200];
set(value(myfig), 'Position', fig_position);

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], ...
```

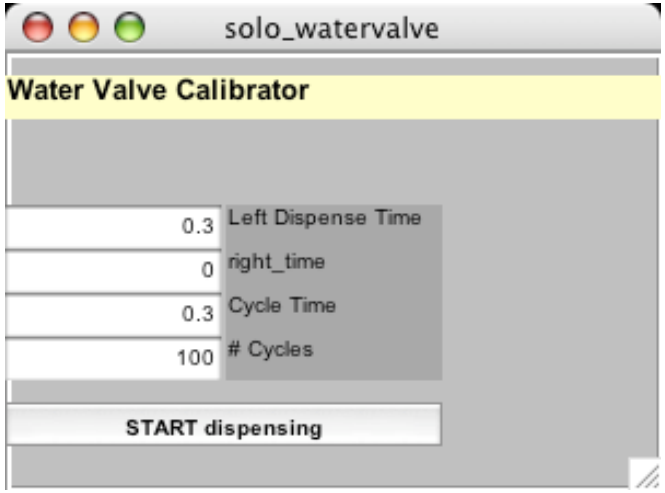


- Optional UI features (labels, tooltips)
- Positioning

Constructor: Add Dispense Control

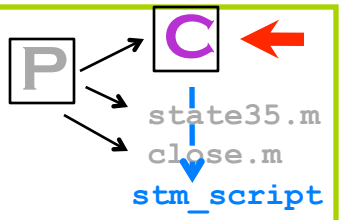



```
next_row(y);
ToggleParam('go', 0, x, y, ...
    'OnString', 'STOP dispensing', ...
    'OffString', 'START dispensing');

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);
```



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));
```

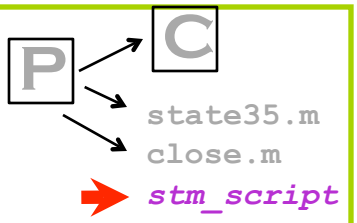
… and calling SoloFunctions

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

Deciding trial sides (*SidesSection.m*)

Constructing sounds (*ChordSection.m*)

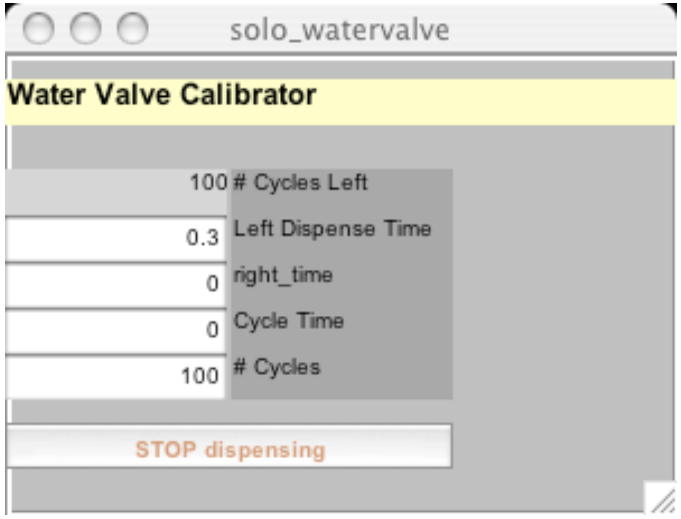
Programming the state matrix: Skeleton for specialised files



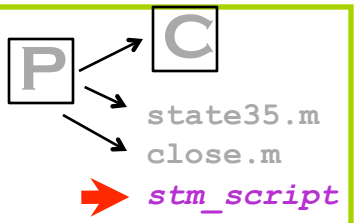

```
function [] = make_and_upload_state_matrix(obj, action, x, y)

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)



```
% 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;
```

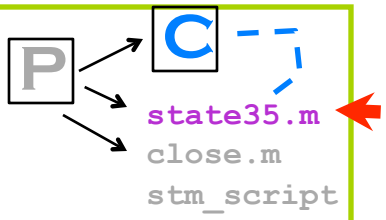


Using read-only args

Using flags set in `mystartup.m`

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

state35.m: Executed at the end of trials



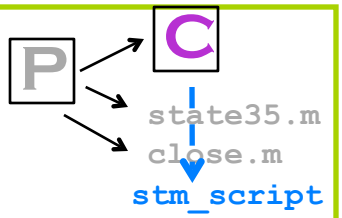
Constructor

```
% ------------------------------------------------------------------
% 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');
```

trial_finished_actions is an SPH!

```
function state35(obj)

GetSoloFunctionArgs;

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

return;
```

Not just a pretty face: Callbacks




```
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'});
```

Constructor: Setting a callback …

```
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;
```

… and executing it.

Not **return**ing after a callback segment => extra state matrix generations!!!

Breathe.
Review the Concepts.

Flow of control

Finding protocol dirs

Interaction between *protocol.m* and *protocol**obj**.m*

End of trial actions (*state35.m*)

Trial update actions (*update.m*)

SoloParamHandles

Creating UI el'ts (EditParam, ToggleParam)

Creating non-UI elements (trial_finished_actions)

Get/set values

Callbacks

Optional parameters (label, tooltipstring)

SoloFunctions

Declaring SoloFunction

Setting r/w variables

Setting read-only vars

GetSoloFunctionArgs !

switch-case stmts

End of Part I

Next Part: A protocol that requires a rat.

Protocol-writing with the Solo system:

Higher, Faster, Stronger:
Localisation Sampling (*Locsamp*)

Bells, whistles, and a few useful things ...

Flow of control

Finding protocol dirs

Interaction between *protocol.m* and *protocol**obj**.m*

End of trial actions (*state35.m*)

Trial update actions (*update.m*)

SoloParamHandles

Creating UI el'ts (EditParam, ToggleParam)

Creating non-UI elements (trial_finished_actions)

Get/set values

Callbacks

Optional parameters (label, tooltipstring)

SoloFunctions

Declaring SoloFunction

Setting r/w variables

Setting read-only vars

GetSoloFunctionArgs !

switch-case stmts

Bells, whistles, and a few useful things …

Flow of control

Finding protocol dirs

Interaction between *protocol.m* and *protocol***obj***.m*

End of trial actions (*state35.m*)

Trial update actions (*update.m*)

Dependencies of SPHs and callbacks

SoloParamHandles

Creating UI el'ts (EditParam, ToggleParam)

Creating non-UI elements (trial_finished_actions)

Get/set values

Callbacks

Optional parameters (label, tooltipstring)

SPH *avatars*: figures, arrays, plots, …

SoloFunctions

Declaring SoloFunction

Setting r/w variables

Setting read-only vars

GetSoloFunctionArgs !

switch-case stmts

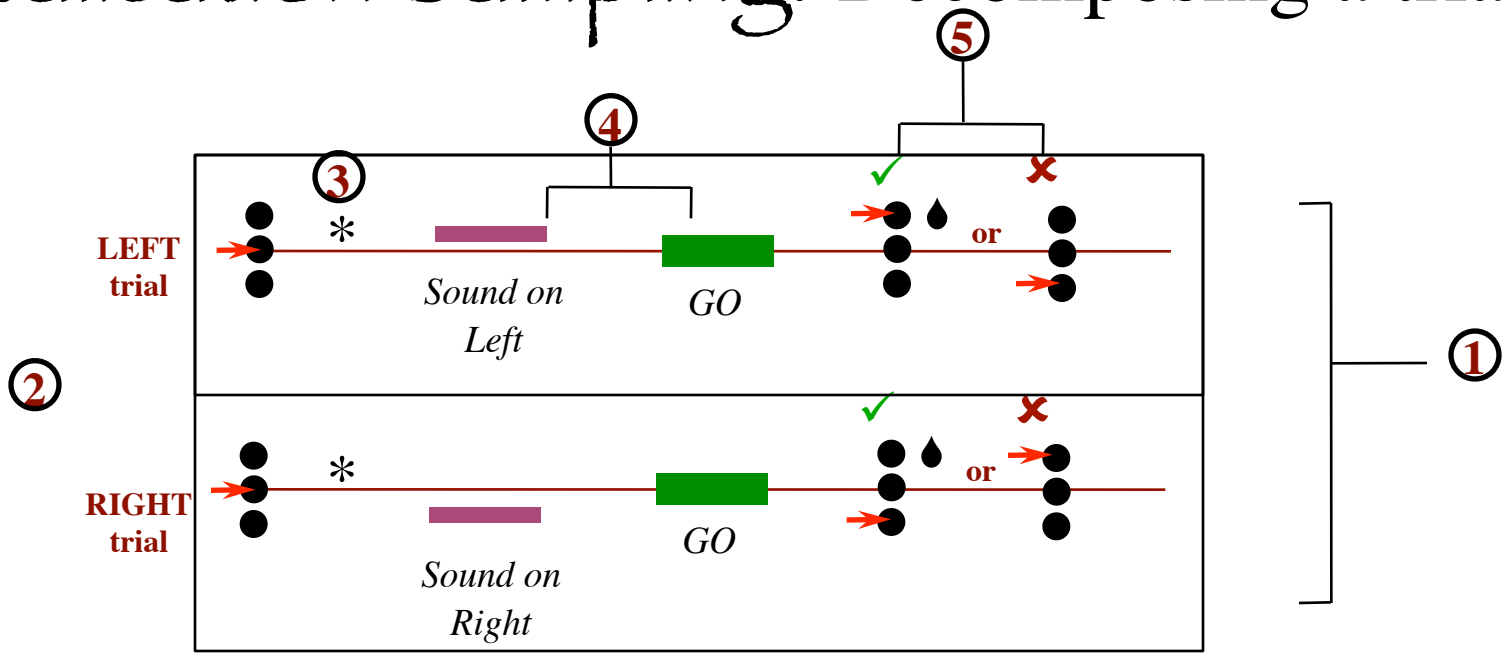
Managing trial data

Saving and Loading

Naming states

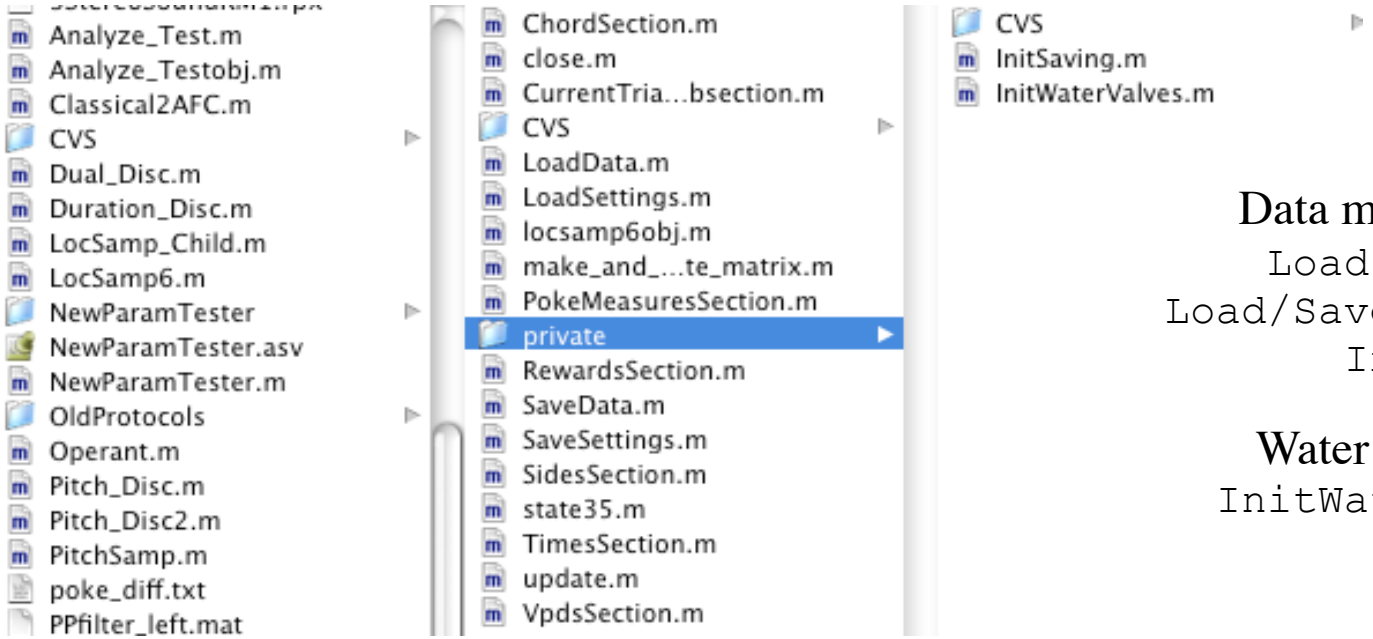
The event cell array

Localisation Sampling: Decomposing a trial



	Functionality	File
1	Trial structure	`make_and_upload_state_matrix.m`
2	Trial sides	`SidesSection.m`
3	Variable Poke Delay	`VpdSection.m`
4	Sounds *(frequency? Duration? Localisation?)*	`ChordSection.m`
5	Tracking correct / incorrect trials	`RewardsSection.m`

Directory view



Data management:
`Load/SaveData`
`Load/SaveSettings`
`InitSaving`

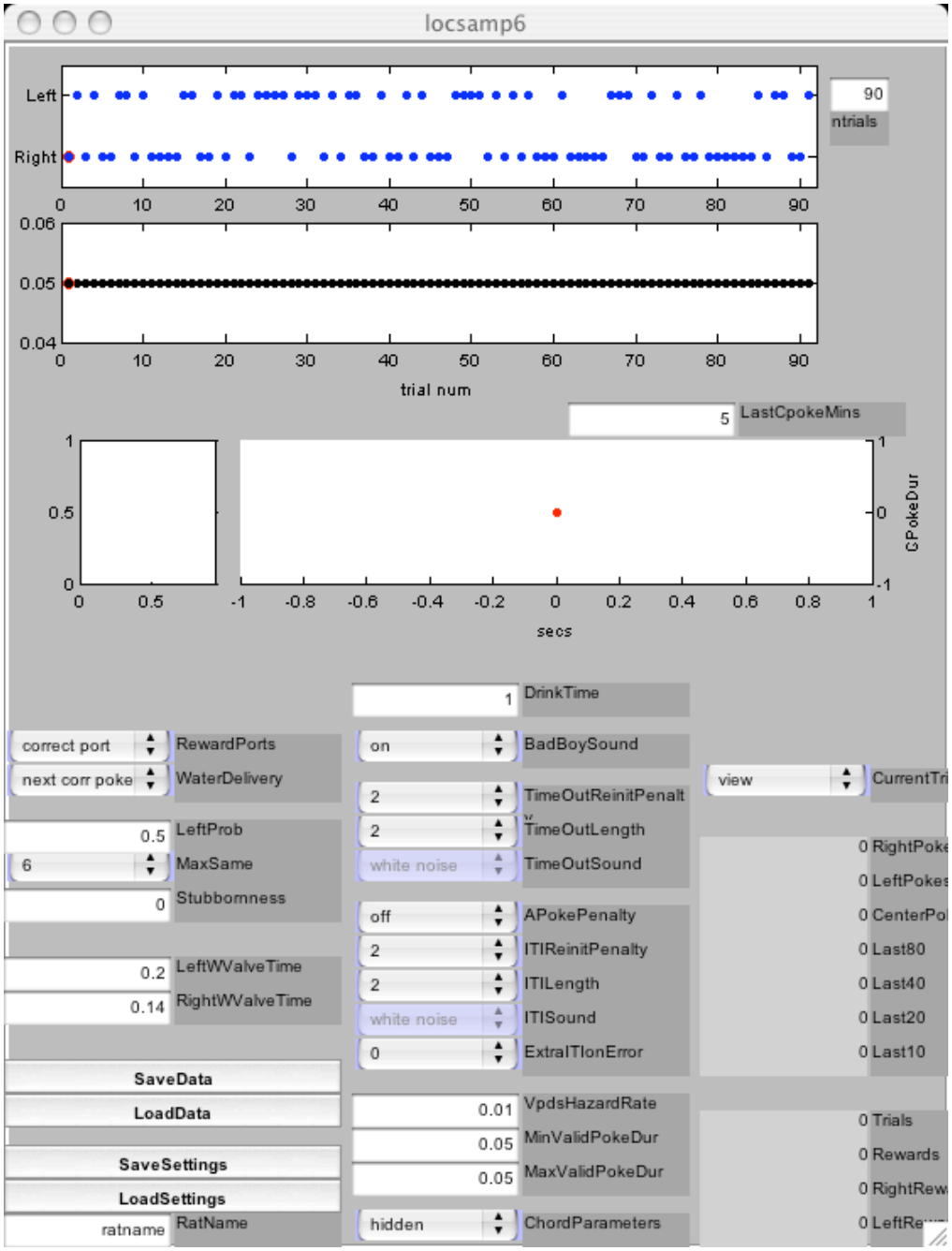
Water dispensing:
`InitWaterValves`

	Functionality	File
1	Trial structure	`make_and_upload_state_matrix.m`
2	Trial sides	`SidesSection.m`
3	Variable Poke Delay	`VpdSection.m`
4	Sounds *(frequency? Duration? Localisation?)*	`ChordSection.m`
5	Tracking correct / incorrect trials	`RewardsSection.m`

- Specialised role within a trial
- Initialise part of main GUI
- Own a set of SPHs
- Pass them to other functions

The main GUI

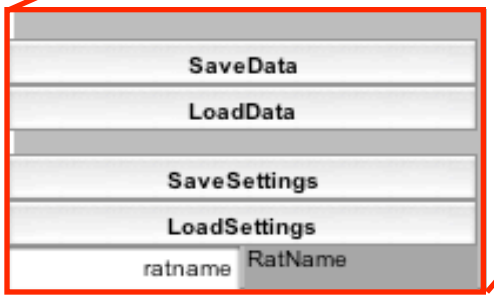
Interaction of many specialised files

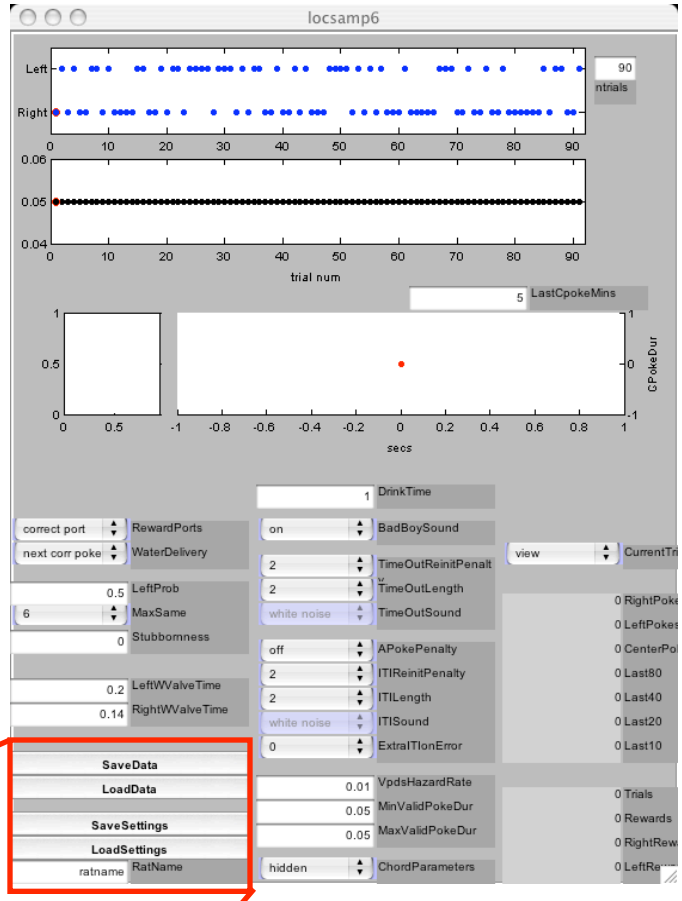


- Specialised role within a trial
- Initialise part of main GUI
- Own a set of SPHs
- Pass them to other functions

Example of a barebones specialised file:

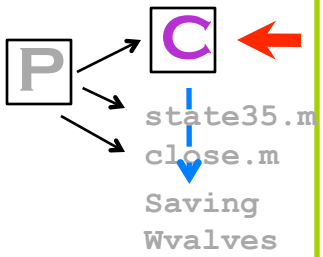
```
function [x, y] = InitSaving(obj, x, y)
%
% [x, y] = InitSaving(x, y)
%
% args:    x, y                    current UI pos, in pixels
%
% returns: x, y                    updated UI pos
%
%

    EditParam(obj, 'RatName', 'ratname', x, y);    next_row(y);
    PushbuttonParam(obj, 'LoadSettings', x, y);    next_row(y);
    PushbuttonParam(obj, 'SaveSettings', x, y);    next_row(y);
    SoloFunction('LoadSettings', 'ro_args', 'RatName');
    SoloFunction('SaveSettings', 'ro_args', 'RatName');
    next_row(y, 0.5);

    PushbuttonParam(obj, 'LoadData', x, y);    next_row(y);
    PushbuttonParam(obj, 'SaveData', x, y);    next_row(y);
    SoloFunction('LoadData', 'ro_args', 'RatName');
    SoloFunction('SaveData', 'ro_args', 'RatName');
    return;
```





- **Specialised role within a trial**
- **Initialise part of main GUI**
- **Own a set of SPHs**
- Pass them to other functions

The constructor:



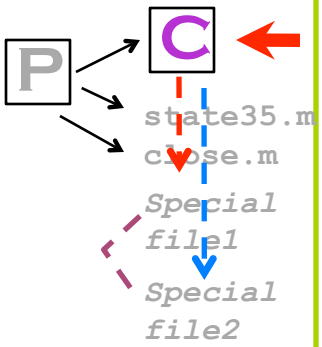
A scaffold for the cascade of dependent SPHs

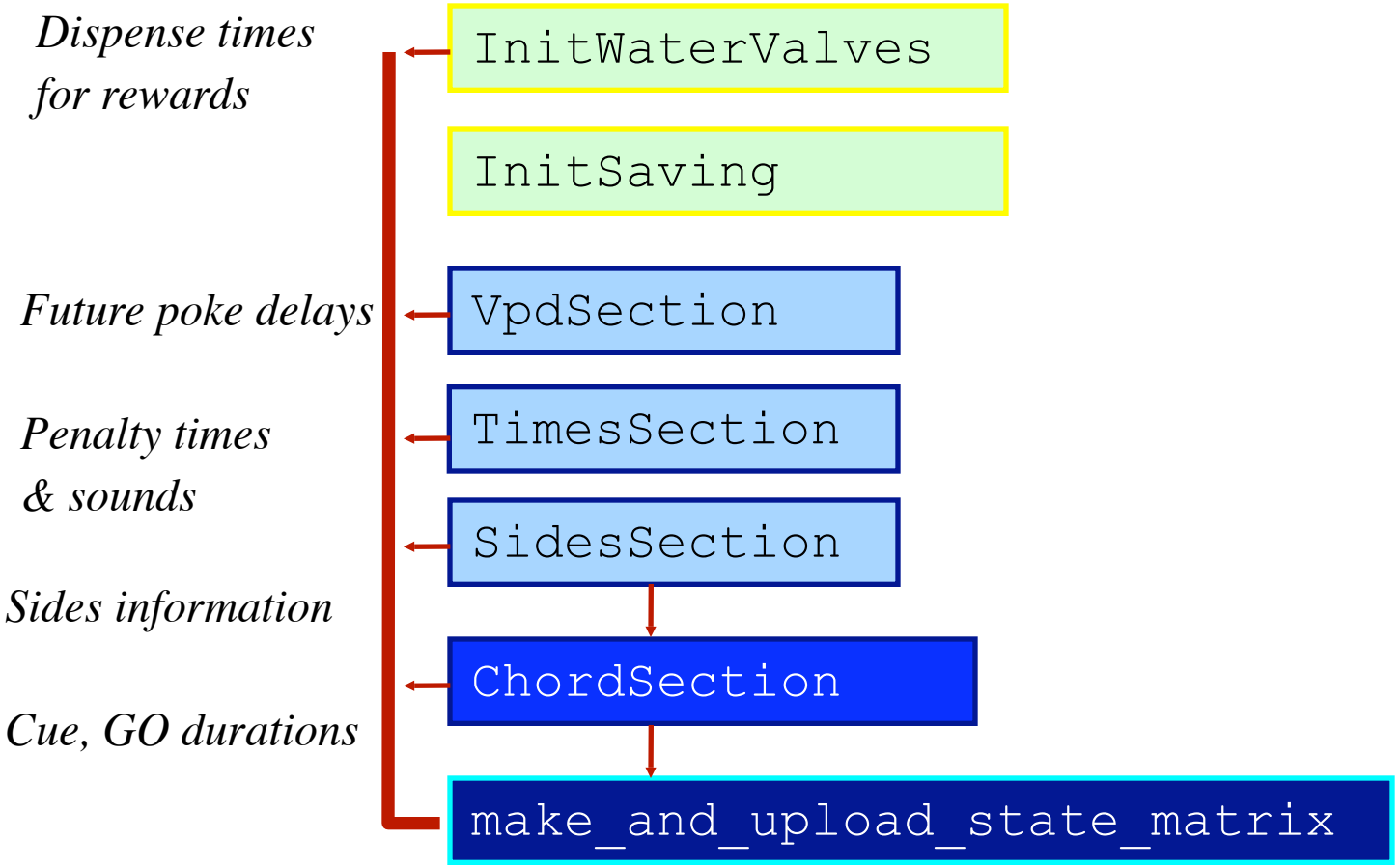
```
% -------- END Magic code that all protocol objects must have ---

set(value(myfig), 'Position', [440 29 559 705])

SoloParamHandle(obj, 'n_done_trials',     'value', 0);
SoloParamHandle(obj, 'n_started_trials', 'value', 0);
SoloParamHandle(obj, 'maxtrials',         'value', 1000);
SoloParamHandle(obj, 'hit_history',       'value', NaN*ones(1, value(maxtrials)));

x = 1; y = 1;                          % Position on GUI
[x, y] =                               InitSaving(obj, x, y);        next_row(y);
[x, y, RightWValve, LeftWValve] = InitWaterValves(obj, x, y);  next_row(y);

SoloFunction('SidesSection', 'ro_args', ...
    {'n_done_trials', 'n_started_trials', 'hit_history','maxtrials'});
[x, y, side_list, WaterDelivery, RewardPorts] = ...
    SidesSection(obj, 'init', x, y); next_row(y, 0.5);
% side_list is a vector of correct sides, one per trial.

next_column(x); y = 1;
SoloFunction('ChordSection', ...
             'ro_args', {'side_list', 'n_done_trials', 'n_started_trials'});
[x, y, chord_sound_len] = ChordSection(obj, 'init', x, y); next_row(y, 0.5);
```

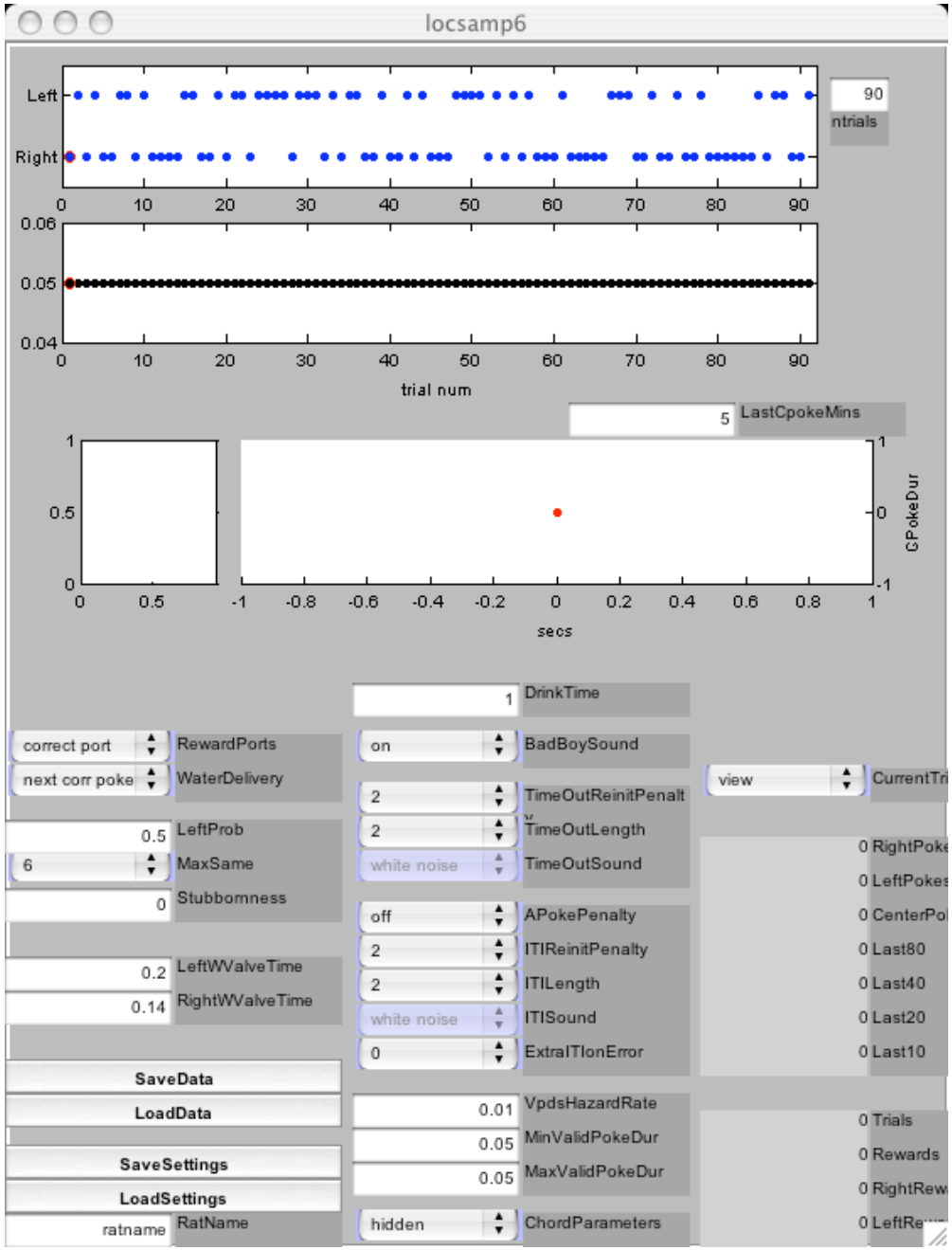
- Specialised role within a trial
- **Initialise part** of main GUI
- **Own a set** of SPHs
- **Pass them to other functions**

The constructor:

A scaffold for the cascade of







Good!

Everything's in

Creative uses of SoloParamHandles

- UI elements

```
EditParam(obj, 'Stubbornness', 0,   x, y); next_row(y);
MenuParam(obj, 'MaxSame', {'1' '2' '3' '4' '5' '6' '7' '8' '
EditParam(obj, 'LeftProb',     0.5, x, y); next_row(y);
```

- Scalars and arrays

  tracking parameters for past trials, setting for future trials

```
SoloParamHandle(obj, 'chord_sound_data');
SoloParamHandle(obj, 'chord_sound_len');
SoloParamHandle(obj, 'side_list', 'value', zeros(1, value(maxtrials)));
```

- Axes

  all plots

- Figures

  reducing workspace clutter

- Any Matlab variables!

  core plot objects (line, patch, text),

Checklist for adding SPH

1. Create

3. Set callbacks

   Call all methods affected by change

5. Update within or after a trial?

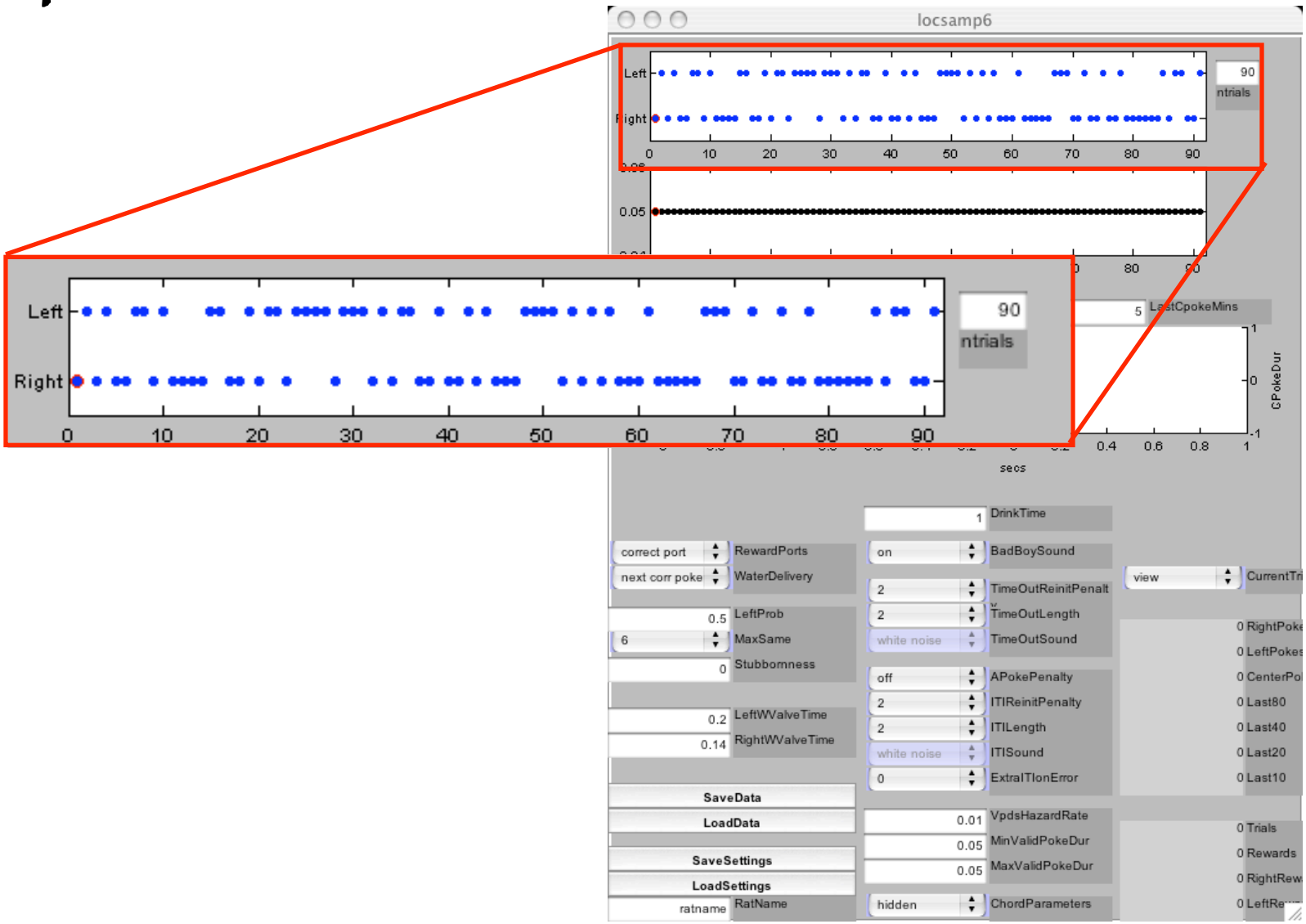
   - Write code for '*update*' action

   (e.g. updating a plot, updating number of successes)

   - Add to *trial_finished_actions* or *trial_update_actions*

Mini-Example 1:
Making and updating

A plot to decide trial sides



1. Create: `SidesSection.m`

Matlab graphic objects contained in SPH:

Axes ←

Line ←

Text ←

```
SoloParamHandle(obj, 'h',   'value', axes('Position', [0.06, 0.88, 0.8, 0.1])); % axes
SoloParamHandle(obj, 'p',   'value', plot(-1, 1, 'b.')); hold on; % blue dots
SoloParamHandle(obj, 'g',   'value', plot(-1, 1, 'g.')); hold on; % green dots
SoloParamHandle(obj, 'r',   'value', plot(-1, 1, 'r.')); hold on; % red dots
SoloParamHandle(obj, 'o',   'value', plot(-1, 1, 'ro')); hold on; % next trial indicator
SoloParamHandle(obj, 'thl','value', text(-ones(1,maxtrials), 0.5*ones(1,maxtrials),'l'));
SoloParamHandle(obj, 'thr','value', text(-ones(1,maxtrials), 0.5*ones(1,maxtrials),'r'));
SoloParamHandle(obj, 'thh','value', text(-ones(1,maxtrials), 0.5*ones(1,maxtrials),'h'));
SoloParamHandle(obj, 'thm','value', text(-ones(1,maxtrials), 0.5*ones(1,maxtrials),'m'));
set_saveable({h;p;g;r;o;thl;thh;thm}, 0);
set([value(thl);value(thr);value(thh);value(thm)], ...
    'HorizontalAlignment', 'Center', 'VerticalAlignment', ...
    'middle', 'FontSize', 8, 'FontWeight', 'bold', 'Color', 'b', ...
    'FontName', 'Helvetica', 'Clipping', 'on');

set(value(h), 'YTick', [0 1], 'YTickLabel', {'Right', 'Left'});
xlabel('');
```

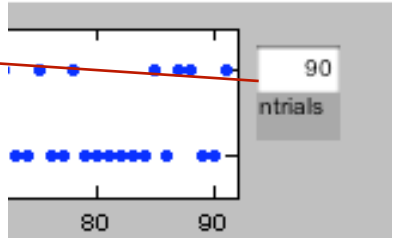
- Specialised role within a trial
- **Initialise part of main GUI**
- **Own a set of SPHs**
- Pass them to other functions

value() call of SPHs of graphic objects returns handles

2. Set callbacks

SidesSection.m

```
% "width", an EditParam to control the # of trials in the plot:
SoloParamHandle(obj, 'width', 'type', 'edit', 'label', 'ntrials', ...
    'labelpos', 'bottom','TooltipString', 'number of trials in plot', ...
    'value', 90, 'position', [490 645 35 40]);
set_callback(width, {'SidesSection', 'update_plot'});
```




```
...

...

...

case 'update_plot', % ----------  UPDATE_PLOT  --------------
    [x, mn, mx] = SidesSection(obj, 'get_width');

    % First, the future:
    set(value(p), 'XData', n_done_trials+1:mx, 'YData', side_
    set(value(h), 'Ylim', [-0.5 1.5], 'XLim', [mn-1 mx+1]);
    set(value(o), 'XData', n_done_trials+1, 'YData', side_lis
    u = n_done_trials;
    if u==0, return; end;

    % Will redraw all points; first clear them off the screen
    set(value(r), 'XData', -1, 'YData', -1);
    set(value(g), 'XData', -1, 'YData', -1);
    % Loop over all done trials:
    for i=1:u,
       % the both-ports-reward trials-- no hit or miss define
       % what matters is just r and l
       if strcmp(get_history(RewardPorts, i), 'both ports'),
          if (side_list(i)==1 & hit_history(i)==1) | ...
                (side_list(i)==0 & hit_history(i)==0),
```

3. Ensure update within/after trial

```
SoloParamHandle(obj, 'trial_finished_actions', 'value', { ...
  'RewardsSection(obj, ''update'');'                        ; ...
  'SidesSection(obj, ''choose_next_side'');'                ; ...
  'SidesSection(obj, ''update_plot'');'                     ; ...
  'VpdsSection(obj, ''update_plot'');'                      ; ...
  'ChordSection(obj, ''make'');'                            ; ...
  'ChordSection(obj, ''upload'');'                          ; ...
  'make_and_upload_state_matrix(obj, ''next_matrix'');'     ; ...
  'CurrentTrialPokesSubsection(obj, ''redraw'')'            ; ...
  'push_history(class(obj));'                               ; ... % no args
});
```

Recall:

trial_finished_actions is processed by **`state35.m`** at the end of every trial

The main protocol file calls **`state35.m`**

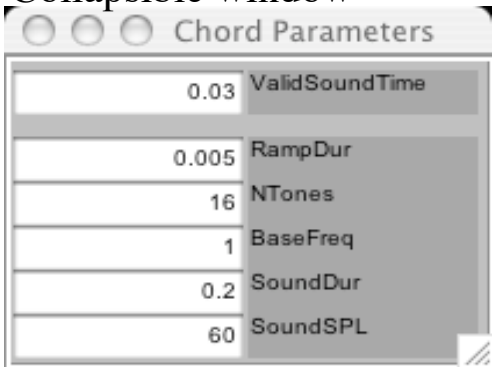
Mini-Example 2:
SPHs that store figure handles

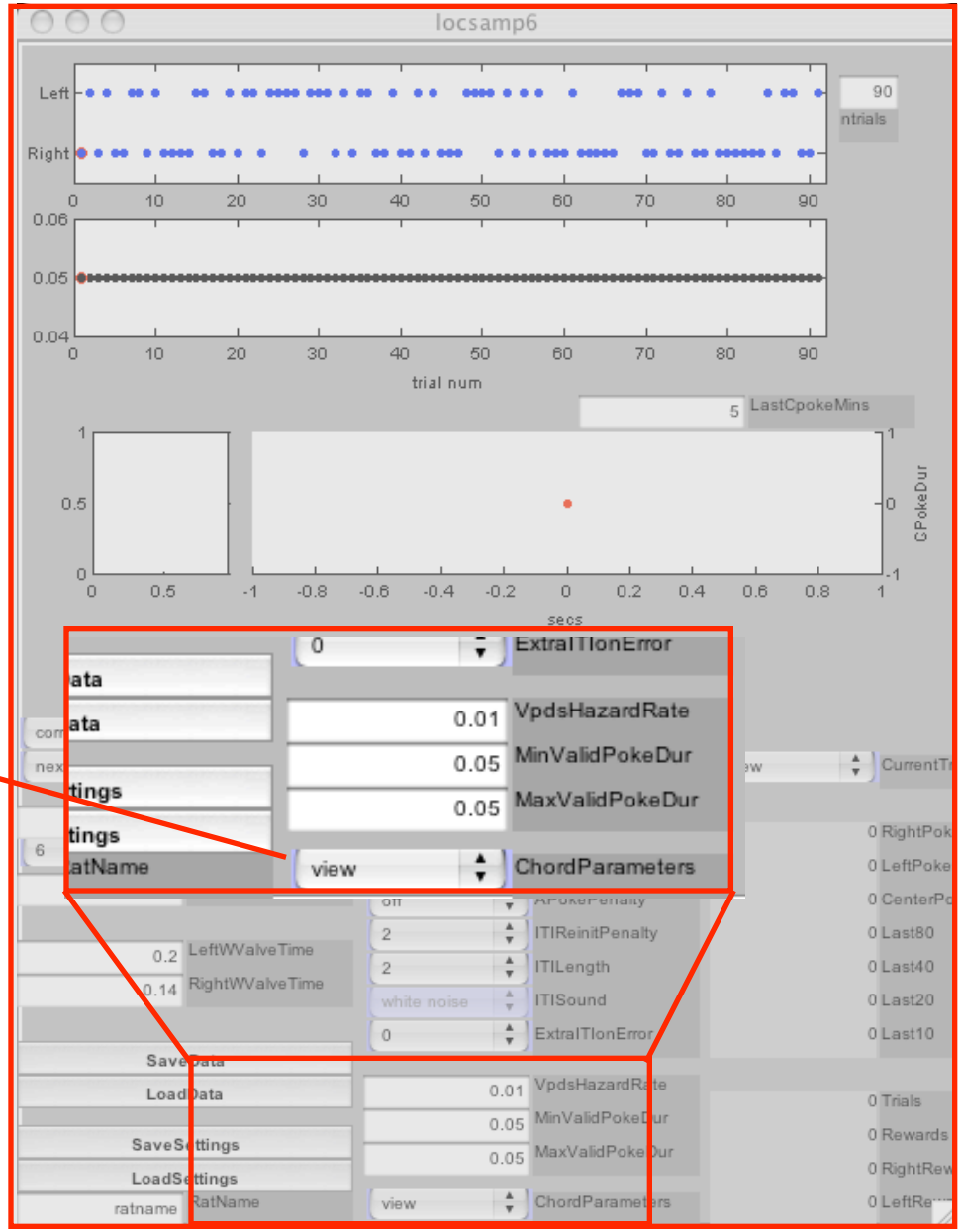
Collapsible windows: A cleaner workspace

`ChordSection.m`

What appears on the main protocol figure

Collapsible window





Making collapsible windows: `ChordSection.m`

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

GetSoloFunctionArgs;
% SoloFunction('ChordSection', 'ro_args', {'side_list', 'n_done_trials'});
% Deals with chord generation and uploading for a protocol.
% Note: This function does not generate the sounds for white-noise (ITI,
% Timeout, etc.,)
% init: Initialises UI parameters specifying types of sound; calls 'make' and 'upload'
% make: Generates chord for the upcoming trial
% upload: The chord is set to be sound type "1" in the RPBox


switch action,
 case 'init'
  fig = gcf; rpbox('InitRP3StereoSound'); figure(fig);

  oldx = x; oldy = y;  x = 5; y = 5;
  SoloParamHandle(obj, 'myfig', 'value', figure, 'saveable', 0);

  EditParam(obj, 'SoundSPL',        60,    x, y);   next_row(y);
  EditParam(obj, 'SoundDur',        0.2,   x, y);   next_row(y);
  EditParam(obj, 'BaseFreq',        1,     x, y);   next_row(y);
  EditParam(obj, 'NTones',          16,    x, y);   next_row(y);
  EditParam(obj, 'RampDur',         0.005, x, y);   next_row(y, 1.5);
  EditParam(obj, 'ValidSoundTime',  0.03,  x, y);   next_row(y);
  set(value(myfig), ...
      'Visible', 'off', 'MenuBar', 'none', 'Name', 'Chord Parameters', ...
      'NumberTitle', 'off', 'CloseRequestFcn', ...
      ['ChordSection(' class(obj) '(''empty''), ''chord_param_hide'')']);
  set_size(value(myfig), [210 141]);
  x = oldx; y = oldy; figure(fig);
  MenuParam(obj, 'ChordParameters', {'hidden', 'view'}, 1, x, y); next_row(y);
  set_callback({ChordParameters}, {'ChordSection', 'chord_param_view'});

  SoloParamHandle(obj, 'chord_sound_data');
  SoloParamHandle(obj, 'chord_sound_len');
                                                    lue', 0);

                                                    , NTones, RampDur, ...
                                                    ion', 'make'});
```

Store fig handle to main figure

Initialise new mini-figure

Adjust its properties

Return control to the main figure after setup

Value(fig) returns a figure handle

"*figure*" makes a figure current

Remember to update coordinates

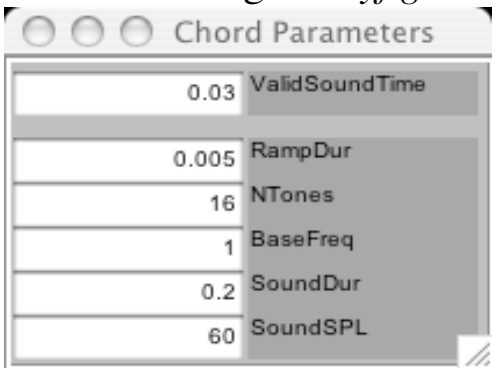
Collapsible windows:

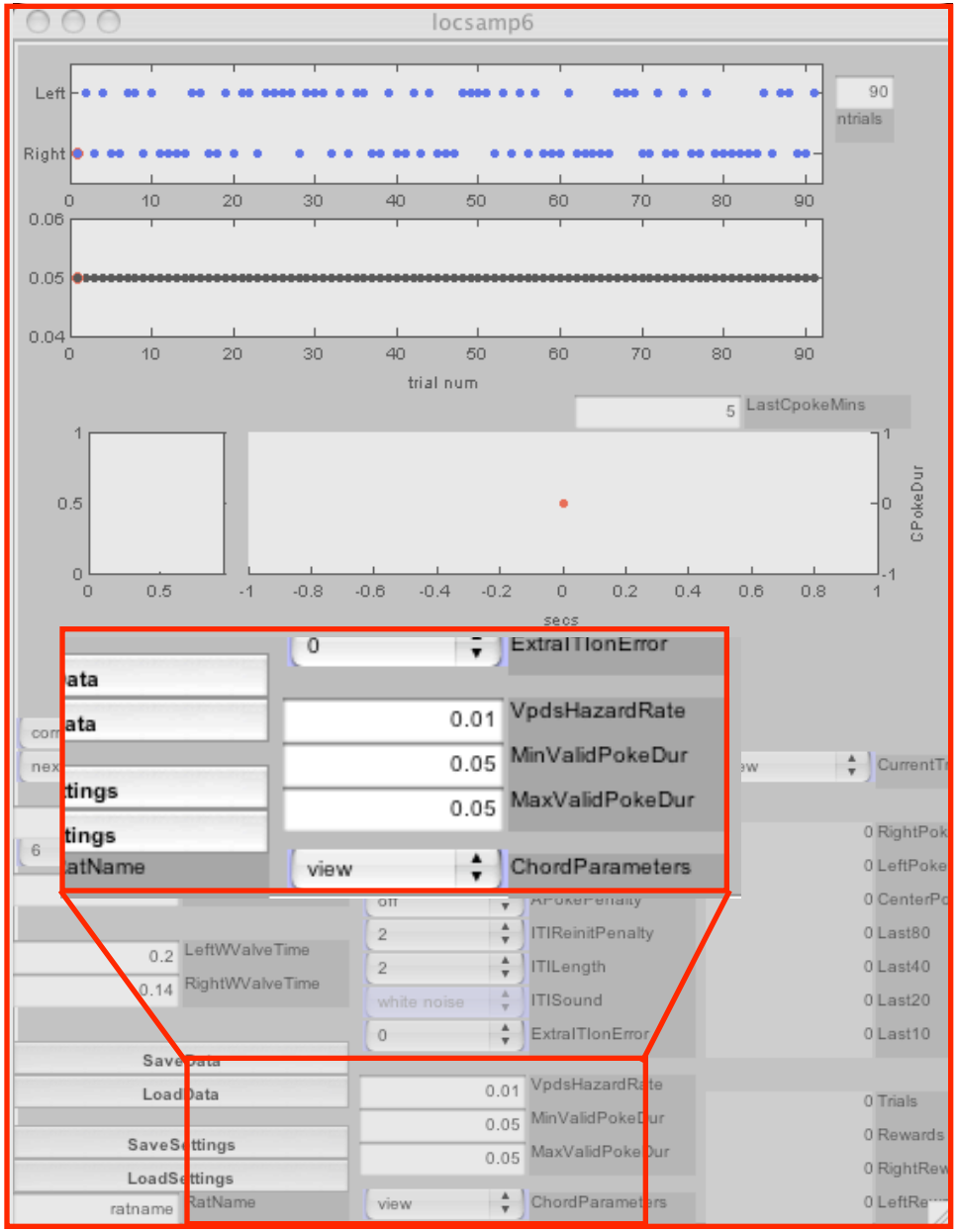
`ChordSection.m`

#1: Create controls in mini-figure

#2: Set callback to hide/show figure in main window

All chord parameters are in figure *myfig*





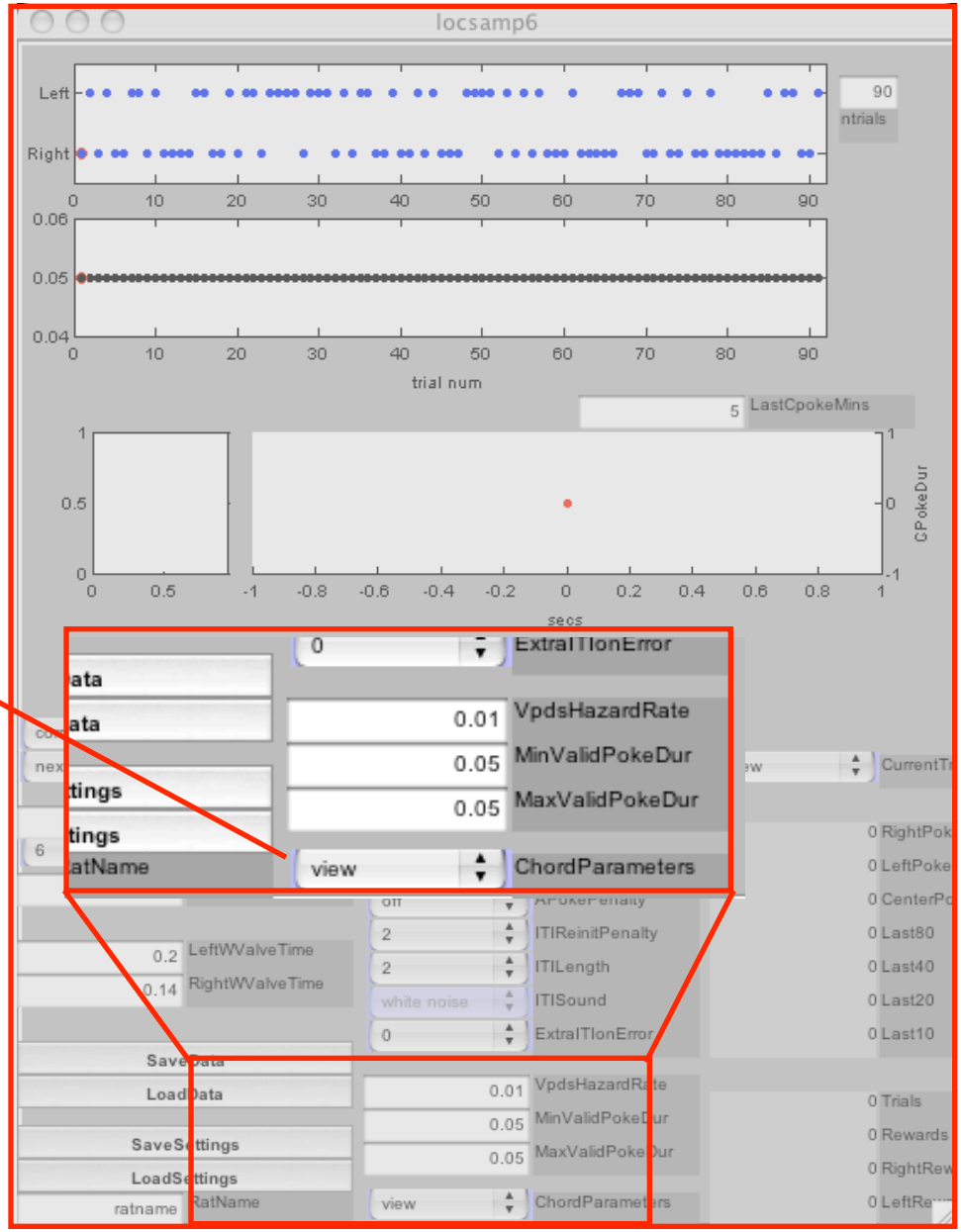
Making collapsible windows: ChordSection.m

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

GetSoloFunctionArgs;
% SoloFunction('ChordSection', 'ro_args', {'side_list', 'n_done_trials'});
% Deals with chord generation and uploading for a protocol.
% Note: This function does not generate the sounds for white-noise (ITI,
% Timeout, etc.,)
% init: Initialises UI parameters specifying types of sound; calls 'make' and 'upload'
% make: Generates chord for the upcoming trial
% upload: The chord is set to be sound type "1" in the RPBox


switch action,
 case 'init'
  fig = gcf; rpbox('InitRP3StereoSound'); figure(fig);

  oldx = x; oldy = y;  x = 5; y = 5;
  SoloParamHandle(obj, 'myfig', 'value', figure, 'saveable', 0);

  EditParam(obj, 'SoundSPL',        60,    x, y);   next_row(y);
  EditParam(obj, 'SoundDur',        0.2,   x, y);   next_row(y);
  EditParam(obj, 'BaseFreq',        1,     x, y);   next_row(y);
  EditParam(obj, 'NTones',          16,    x, y);   next_row(y);
  EditParam(obj, 'RampDur',         0.005, x, y);   next_row(y, 1.5);
  EditParam(obj, 'ValidSoundTime',  0.03,  x, y);   next_row(y);
  set(value(myfig), ...
      'Visible', 'off', 'MenuBar', 'none', 'Name', 'Chord Parameters', ...
      'NumberTitle', 'off', 'CloseRequestFcn', ...
      ['ChordSection(' class(obj) '(''empty''), ''chord_param_hide'')']);
  set_size(value(myfig), [210 141]);
  x = oldx; y = oldy; figure(fig);
  MenuParam(obj, 'ChordParameters', {'hidden', 'view'}, 1, x, y); next_row(y);
  set_callback({ChordParameters}, {'ChordSection', 'chord_param_view'});

  SoloParamHandle(obj, 'chord_sound_data');
  SoloParamHandle(obj, 'chord_sound_len');
  SoloParamHandle(obj, 'chord_uploaded', 'value', 0);

  set_callback({SoundSPL, SoundDur, BaseFreq, NTones, RampDur, ...
                ValidSoundTime}, {'ChordSection', 'make'});

  ChordSection(obj, 'make');
  ChordSection(obj, 'upload');
```

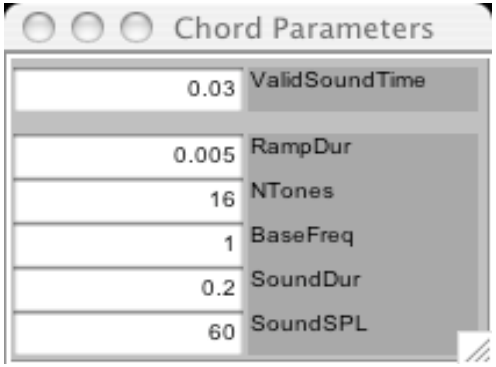
Collapsible windows: A cleaner workspace

`ChordSection.m`

What appears on the main protocol figure



All initialised chord parameters are in the *myfig* figure



Bells, whistles, and a few useful things ...

Flow of control

Finding protocol dirs

Interaction between *protocol.m* and *protocol**obj**.m*

End of trial actions (*state35.m*)

Trial update actions (*update.m*)

SoloParamHandles

Creating UI el'ts (EditParam, ToggleParam)

Creating non-UI elements (trial_finished_actions)

Get/set values

Callbacks

Optional parameters (label, tooltipstring)

SoloFunctions

Declaring SoloFunction

Setting r/w variables

Setting read-only vars

GetSoloFunctionArgs !

switch-case stmts

Managing trial data

Saving and Loading

Making and sending sounds

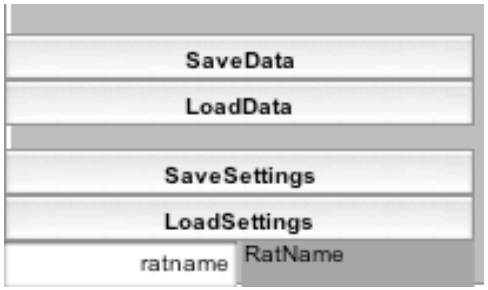
Naming states

The event cell array

Loading and Saving: UI elements

InitSaving.m

```
function [x, y] = InitSaving(obj, x, y)

EditParam(obj, 'RatName', 'ratname', x, y);   next_row(y);
PushbuttonParam(obj, 'LoadSettings', x, y);   next_row(y);
PushbuttonParam(obj, 'SaveSettings', x, y);   next_row(y);
SoloFunction('LoadSettings', 'ro_args', 'RatName');
SoloFunction('SaveSettings', 'ro_args', 'RatName');
next_row(y, 0.5);

PushbuttonParam(obj, 'LoadData', x, y);   next_row(y);
PushbuttonParam(obj, 'SaveData', x, y);   next_row(y);
SoloFunction('LoadData', 'ro_args', 'RatName');
SoloFunction('SaveData', 'ro_args', 'RatName');
return;
```



Scripts to load/save data/settings:

`*_soloparamvalues.m`: Loads/saves data

`*_solo`**`ui`**`paramvalues.m`: Loads/saves settings

`*` = `load` or `save`

Loading …

LoadData.m

```
function [] = LoadData(obj);

    GetSoloFunctionArgs;

    load_soloparamvalues(RatName);

    SidesSection(obj,          'set_future_sides');
    SidesSection(obj,          'update_plot');
    VpdsSection(obj,           'set_future_vpds');
    VpdsSection(obj,           'update_plot');
    PokeMeasuresSection(obj,   'update_plot');
    ChordSection(obj,          'make');
```

LoadSettings.m

```
function [] = LoadSettings(obj);

    GetSoloFunctionArgs;

    load_solouiparamvalues(RatName);

    SidesSection(obj, 'set_future_sides');
    SidesSection(obj, 'update_plot');
    VpdsSection(obj, 'set_future_vpds');
    VpdsSection(obj, 'update_plot');
    ChordSection(obj, 'make');
```

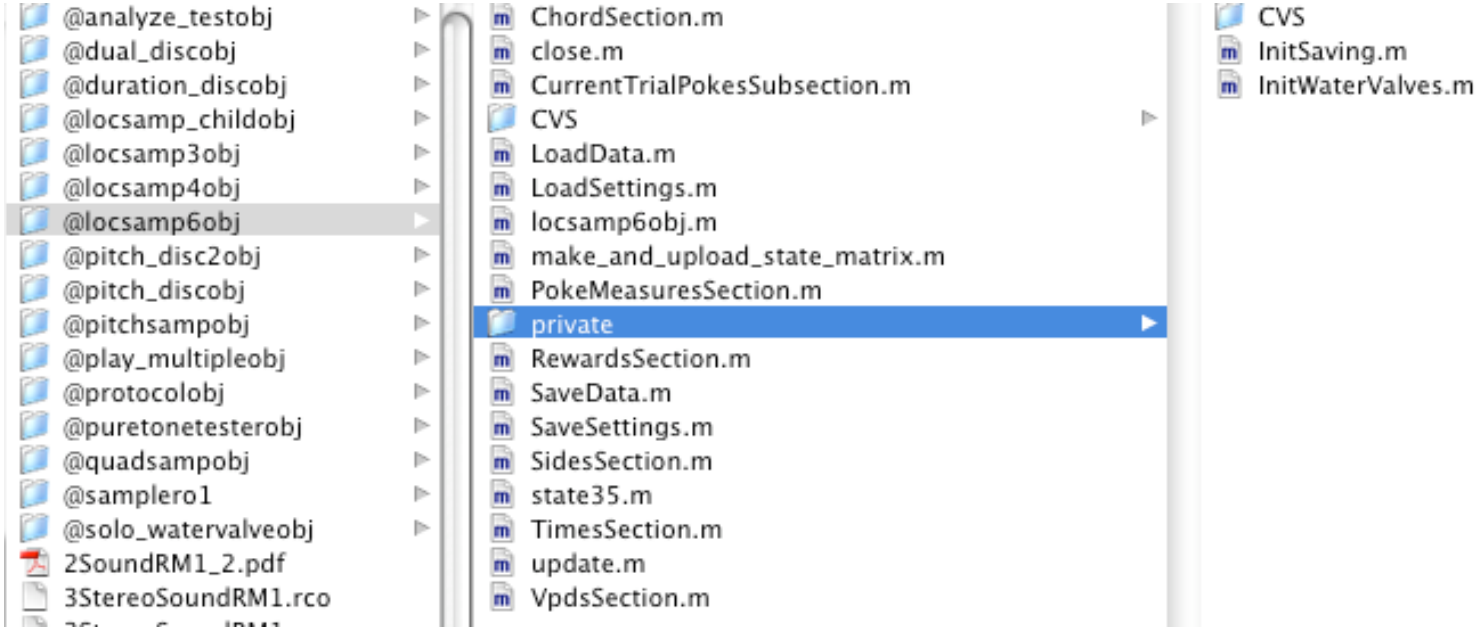
Since SPH values have changed,
make a callback!

… and saving

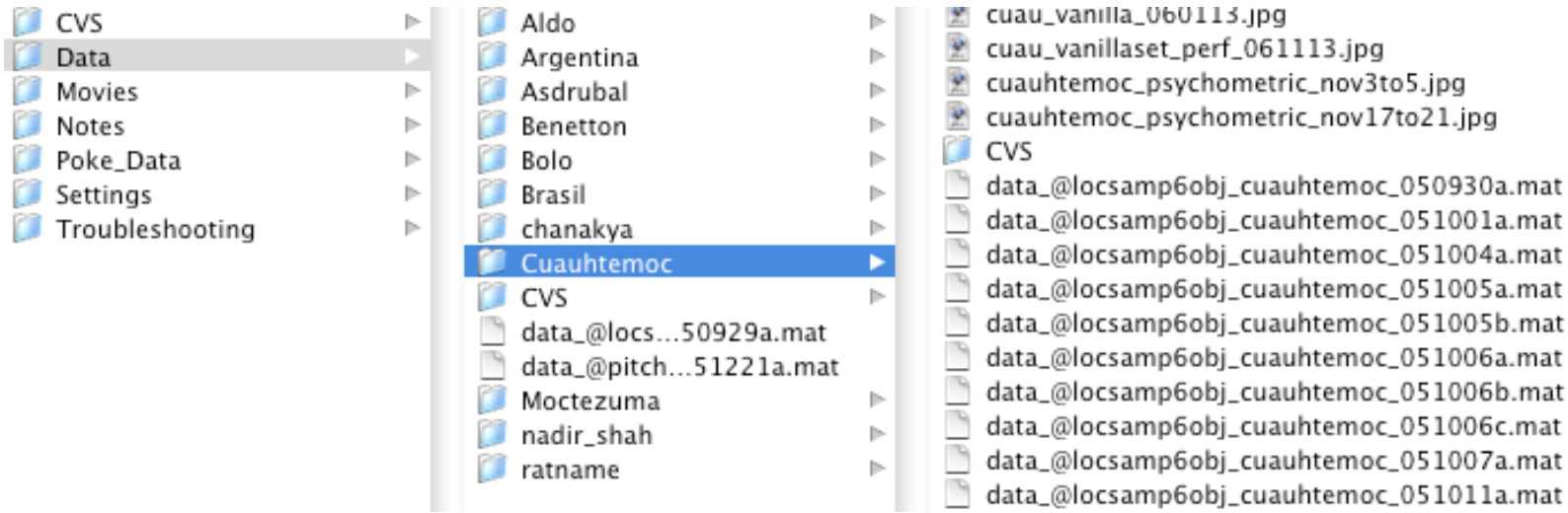
SaveData.m

```
function [] = SaveData(obj);

    GetSoloFunctionArgs;

    save_soloparamvalues(RatName);
```

Recap: Directory view



Using data files



`load_datafile`(*ratname*, *taskname*, *date*) returns:

- *saved_history*: cell arrays of trial-by-trial settings
- *saved:* arrays that have information across all trials (e.g. *side_list*)
- *fig_position*: Position of the main protocol figure
- *saved_autoset*: strings of autoset commands

Path to saved files: `(root-dir)/SoloData/`

Data files: `Data/(ratname)/data_taskname_datef.mat`

Settings files: `Settings/(ratname)/settings_taskname_datef.mat`

Using data files (cont'd)

- Naming convention:

  data_struct.*SoloFunction_name_SPH_name*

  e.g. saved.SidesSection_side_list

  e.g. saved_history.make_and_upload_state_matrix_badboy_sound

```
>> evs = saved_history.quadsampobj_LastTrialEvents;
>> whos evs
  Name      Size                    Bytes  Class

  evs     101x1                    384444  cell array

Grand total is 47399 elements using 384444 bytes

 >> e = evs{5}; e(100:110,:)

 ans =

    1.0e+03 *

     0.0400         0    1.4736
     0.0400    0.0050    1.4768
     0.0400    0.0060    1.4768
     0.0400    0.0010    1.4821
     0.0410         0    1.4821
     0.0410         0    1.4821
     0.0420         0    1.4821
     0.0420    0.0020    1.4821
     0.0430         0    1.4821
     0.0430         0    1.4821
     0.0440         0    1.4821
```

Example #1: Snapshots of event history of trial #5

```
>> sides = saved.SidesSection_side_list; sides(5:10)

ans =

     1     0     1     1     0     0
```

Example #2: Trial sides for trials #5-10

How is trial information stored?

1. Values of all SPHs : *push_history*
2. State matrix: *RealTimeStates*
3. Events during the trial: *LastTrialEvents*

push_history: Storing SPH values on a trial-by-trial basis

Pop Quiz:
You want to store SPH values on a trial-by-trial basis. Now what?

Hint: The call to make is *push_history(class(obj))*.
The question is, where?

Mystery to be solved during the tutorial …

RealTimeStates: Mapping names to state numbers

Setting: **`make_and_upload_state_matrix.m`**

```
%       Cin     Cout    Lin     Lout    Rin     Rout    Tup     Timer   D
stm = [stm ; ...
    1+b     b       b       b       b       b       b       100     0
    1+b     b       b       b       b       b       2+b     0.01    0
    TouS    TouS    TouS    TouS    TouS    TouS    3+b     prst    0
    3+b     3+b     3+b     3+b     3+b     3+b     4+b     vlst    0
    0       0       0       0       0       0       ptnA    lost    0
    WpkS    WpkS    lpkA    WpkS    rpkA    WpkS    WpkS    100     0
    WtoS    WtoS    WtoS    WtoS    WtoS    WtoS    TouS    tdur    0
    ];
end;


RealTimeStates.pre_chord = pstart + 2;
RealTimeStates.chord     = pstart + (3:4);
```

… and what it looks like:

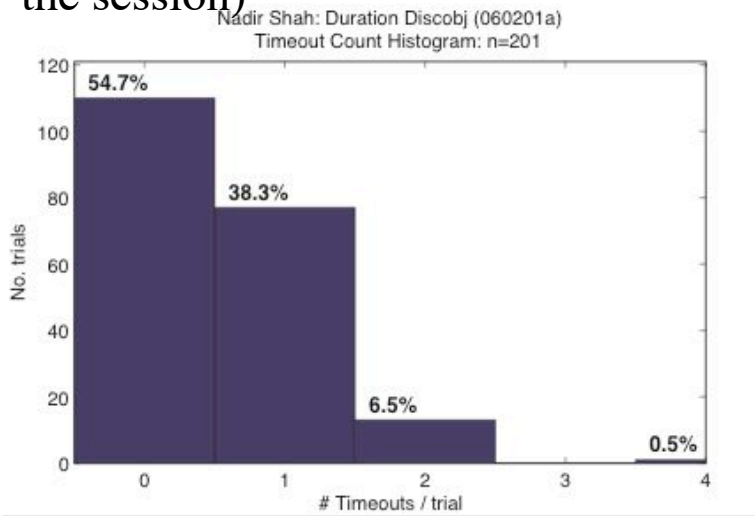
```
wait_for_cpoke: 40
     pre_chord: [42 43 44 45 46 47 48 49 50 51 52 53 54]
         chord: [257 258 259 260 261 262]
wait_for_apoke: 263
   left_dirdel: 268
  right_dirdel: 270
   left_reward: 264
  right_reward: 266
     drink_time: [265 267]
        timeout: [286 287 288 289 290 291 292 293]
            iti: [274 275 276 277 278 279]
      dead_time: [1 2 3 4 5 6 7 8 9 10 35]
        state35: 36
      extra_iti: [280 281 282 283 284 285]
            cue: [1x157 double]
         pre_go: [1x45 double]
```

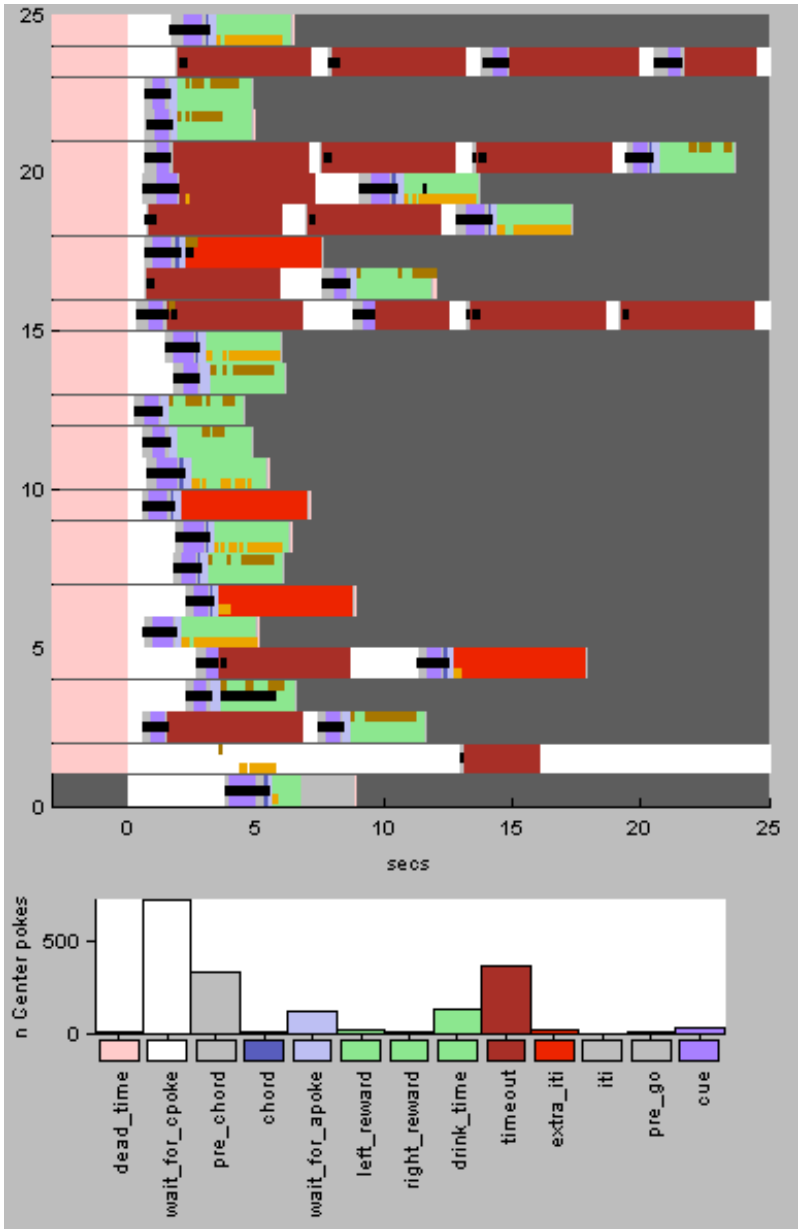
RealTimeStates:

a struct (sets of key-value pairs)
Key: state name
Value: Corresponding state numbers (vector)

Examples of uses for *RealTimeStates*: Mapping names to state numbers

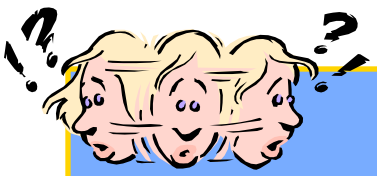
Easier time building analysis tool:
(e.g. Distribution of # timeouts across the session)





Extensions: Colour-code states to monitor current and past activity during the session

LastTrialEvents: Storing event history



Pop Quiz:

The pokes plot (previous page) updates *during* a trial. The pokes plot uses *LastTrialEvents*. How would you find out where *LastTrialEvents* is updated?

Mystery to be solved during the tutorial …

LastTrialEvents: Storing event history

Structure: Matrix of *e* rows and 3 columns,

e : Number of events during the trial

Columns:

1) State number in which event occurred
2) Event number (default: 1-Cin, 2-Cout, 3-Lin, etc.,)
3) Time from session start

```
>> evs = saved_history.quadsampobj_LastTrialEvents;
>> whos evs
  Name      Size                    Bytes  Class

  evs     101x1                    384444  cell array

Grand total is 47399 elements using 384444 bytes

 >> e = evs{5}; e(100:110,:)

 ans =

    1.0e+03 *

     0.0400         0    1.4736
     0.0400    0.0050    1.4768
     0.0400    0.0060    1.4768
     0.0400    0.0010    1.4821
     0.0410         0    1.4821
     0.0410         0    1.4821
     0.0420         0    1.4821
     0.0420    0.0020    1.4821
     0.0430         0    1.4821
     0.0430         0    1.4821
     0.0440         0    1.4821
```

Snapshots of event history of trial #5

Inhale. Exhale.
Repeat.

Reviewing Concepts

Core concepts: SoloWaterValve

Flow of control

Finding protocol dirs

Interaction between *protocol.m* and *protocol**obj**.m*

End of trial actions (*state35.m*)

Trial update actions (*update.m*)

SoloParamHandles

Creating UI el'ts (EditParam, ToggleParam)

Creating non-UI elements (trial_finished_actions)

Get/set values

Callbacks

Optional parameters (label, tooltipstring)

SoloFunctions

Declaring SoloFunction

Setting r/w variables

Setting read-only vars

GetSoloFunctionArgs !

switch-case stmts

Layer II: Localisation sampling

Flow of control

Finding protocol dirs

Interaction between *protocol.m* and *protocolobj.m*

End of trial actions (*state35.m*)

Trial update actions (*update.m*)

SoloParamHandles

Creating UI el'ts (EditParam, ToggleParam)

Creating non-UI elements (trial_finished_actions)

Get/set values

Callbacks

Optional parameters (label, tooltipstring)

SoloFunctions

Declaring SoloFunction

Setting r/w variables

Setting read-only vars

GetSoloFunctionArgs !

switch-case stmts

Managing trial data

Saving and Loading

Making and sending sounds

Naming states

The event cell array