NIMH MonkeyLogic 2

Jaewon Hwang
Staff Scientist
Laboratory of Neuropsychology
National Institute of Mental Health, NIH

NIMH MonkeyLogic 2

- NIMH MonkeyLogic (Feb 17, 2017 or earlier) inherited most of its code from the original MonkeyLogic (Oct 2014) and was built by patching the old code.

  → The UI was not organized well and did not reflect the newly added features.

  → Some issues due to the structure of the software were difficult to fix.
  (lack of support for multiple DAQ boards, other datafile formats, etc.)

- NIMH MonkeyLogic 2 (ML2) is re-written from scratch in the object-oriented programming style.

  → The size of the code is reduced significantly while ML2 supports more functions and new features.

  → All codes are modularized. The extensibility of the code is improved and the maintenance is easier.

System Requirement

- MATLAB R2011a or later
  - 32-bit or 64-bit
  - No additional toolbox is required

- Windows 7 or later

- Microsoft Visual C++ 2013 Redistributable†
  (https://www.microsoft.com/en-us/download/details.aspx?id=40784)

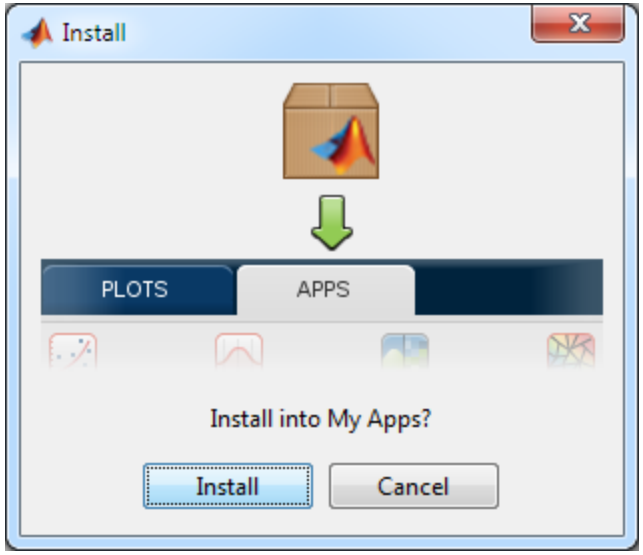
- DirectX 9.0c runtime†
  (https://www.microsoft.com/en-us/download/details.aspx?id=8109)

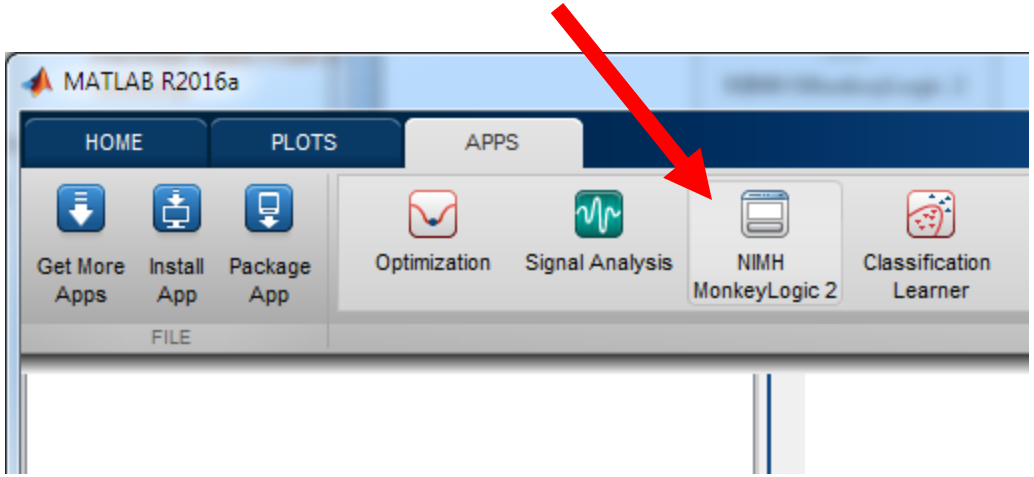
- Currently the products of National Instruments are the only DAQ devices supported.

† If these libraries are not installed, users will be led to the download website upon starting MonkeyLogic 2.

Installation

- In addition to the traditional ZIP package, NIMH MonkeyLogic 2 is distributed as a MATLAB app.
- One click installation/uninstallation
- No other setting (path & directory setup) is necessary.





New Features in NIMH MonkeyLogic 2 (1/3)

- Support for multiple DAQ boards
  This feature differs from the duplicate board setting of the original MonkeyLogic. If you have many DAQ boards, you can assign analog channels from any of those boards. They don't have to be all on one board. This is particularly useful for simultaneous stimulations via multiple AO devices. Also analog buttons and digital buttons.

- The number of the I/O channels that can be assigned to buttons, TTLs and general inputs has already increased in NIMH MonkeyLogic. In NIMH MonkeyLogic 2, users can add more of them simply by editing mliolist.m

- New file format: BHV2 (private binary format) and H5 (HDF5 format)

- The timing code change is registered correctly without restarting MonkeyLogic in the old versions of MATLAB.

- The values of "editable" variables are kept in the configuration file, so users do not need to re-type them every time a new session starts.

New Features in NIMH MonkeyLogic 2 (2/3)

- 'alert_function.m' is called when a new task starts/ends, when a new block starts/ends and when a new trial starts/ends. Users can use this function to notify the progress of the experiment or change the behavior of the timing file code.

- User can keep their customized copies of codes.txt, reward_function.m and alert_function.m in the task directory with the conditions and timing files. NIMH MonkeyLogic 2 reads them from the task directory first, if the files with the same names exist both in the task directory and the MonkeyLogic base directory.

- The minimum ITI required for data saving and stimulus creation in NIMH MonkeyLogic 2 is much shorter than that in the old MonkeyLogic.

- All time records are stored without rounding off so that microsecond precision can be preserved.

New Features in NIMH MonkeyLogic 2 (2/3)

- In the previous ML versions, the reaction time (RT) that eyejoytrack() returned was not calculated from the screen flip time of toggleobject(), but from the beginning of eyejoytrack(). Therefore, the RT was always shorter than the actual value. To provide a better measure, now eyejoytrack() returns a third argument which is a trialtime() reading when the target behavior is detected.

```
fliptime = toggleobject(1);
[ontarget, rt, time_on_response] = eyejoytrack('acquirefix',1,3,1000);
better_rt = time_on_response - fliptime;
```

- Now goodmonkey() takes the 'eventmarker' option so that the beginning of the reward delivery can be stamped in the neural data.

```
goodmonkey(100, 'NumReward',3, 'eventmarker',99);  % mark the timestamp of each of 3 reward drops
```

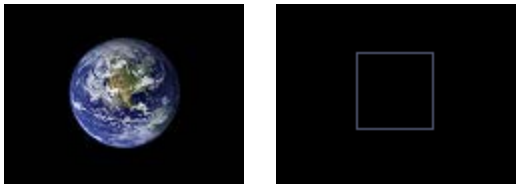
- Stimulus presentation and event-marking in toggleobject are processed at the DLL level to minimize the latency between the screen flip time and the event timestamp.

Features Introduced in NIMH MonkeyLogic 1 (1/3)

- NIMH DAQ Toolbox enables MonkeyLogic to run on both 32-bit and 64-bit MATLAB and performs near-realtime behavior monitoring without requiring duplicate DAQ boards.

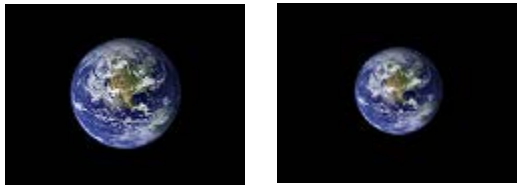
- "What you see is what your monkey sees." The control screen displays the same scene as the subject screen, instead of diagrammatic forms of objects.

Old MonkeyLogic



Subject screen **Control screen**

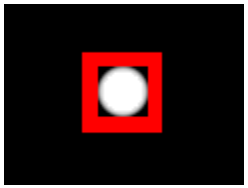
NIMH MonkeyLogic



Subject screen **Control screen**

- Support for transparent images by both alpha blending and color keying

Old MonkeyLogic



NIMH MonkeyLogic

Features Introduced in NIMH MonkeyLogic 1 (2/3)

- Support for mouse/touchscreen and USB joysticks
  (Some touchscreens that do not translate touches into mouse messages are not compatible.)

- Low-latency audio output support (as short at 25 msec in private tests), using Microsoft XAudio2 APIs.

- Support for movie streaming. There is no limit in the movie length that can be played.

- The computer does not go to sleep or turn off the screens while a task is running.

- A new calibration method that does not require Image Processing Toolbox

- NIMH MonkeyLogic can be run with just one monitor and does not require a DAQ board during the simulation mode. This makes it possible to test new tasks without lab equipment.

Features Introduced in NIMH MonkeyLogic 1 (3/3)

- Each button is separately simulated by pressing the key 1-9 and 0. The button input is on while users hold the key press and becomes off as soon as the key is released, just like a real button.

- The “AbsoluteTrialStartTime” field in the BHV file stores the elapsed time from the start of Trial 1 in milliseconds. This is measured with tic/toc and more reliable than the system clock reading. The clock time checked at the beginning of each trial is stored in the “TrialDateTime” field instead.

- User can manually correct eye drift by pressing the ‘C’ key during eyejoytrack() in the task. The ‘U’ key undoes the previous drift correction. You can do/undo multiple times. This manual correction works even when you are using the raw signals without calibration.

User Interface (1/4)

1. Centralized display for all error messages and notifications.

2. Online manual for the conditions file

3. Edit button for the selected conditions file

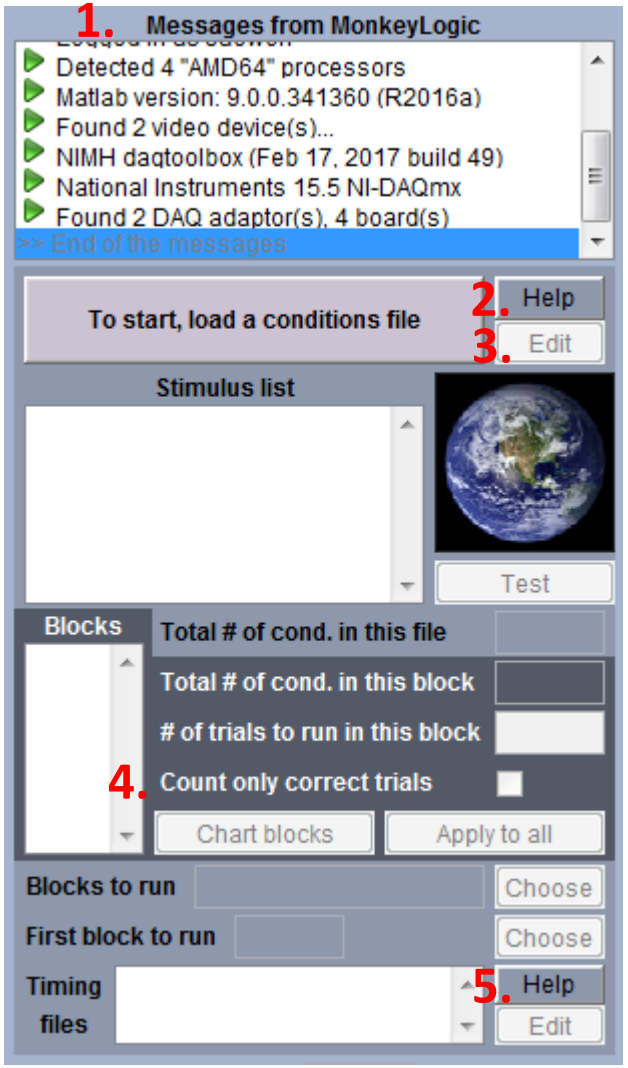
4. Whether to count only correct trials can be set for each block individually.

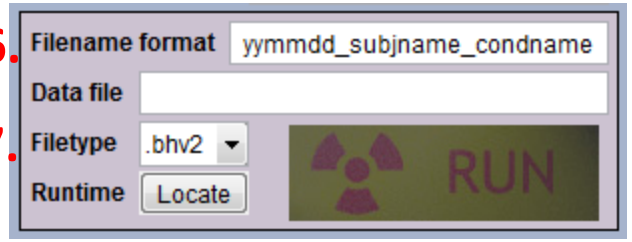
5. Button to the timing file online manual

6. Customizable default data filename

7. New data format: bhv2 and h5

- Both formats save/restore workspace variables as they are. There is no need to modify the code when the data structure changes.
- bhv2 is a private binary format based on a simple recursive algorithm. Although it is not supported in any other software, you can easily write your own reader.
- h5 is a file extension for the HDF5 format. HDF5 is supported by many commercial and non-commercial software, but it produces a bit larger files than bhv2 and the I/O speed is also slower (especially for reading).
- mlread.m included in MonkeyLogic 2 provides one single read interface for both formats. Type help mlread on the MATLAB command window for details.





User Interface (2/4)

8. Pop-up tooltips on most of the UI components

9. Latency test that does not require hardware configuration

10. Fallback window that can substitute for the subject screen

- The contents of the subject screen can be presented in a window so that MonkeyLogic can be run just with one monitor.

11. The use of the fallback window can be forced

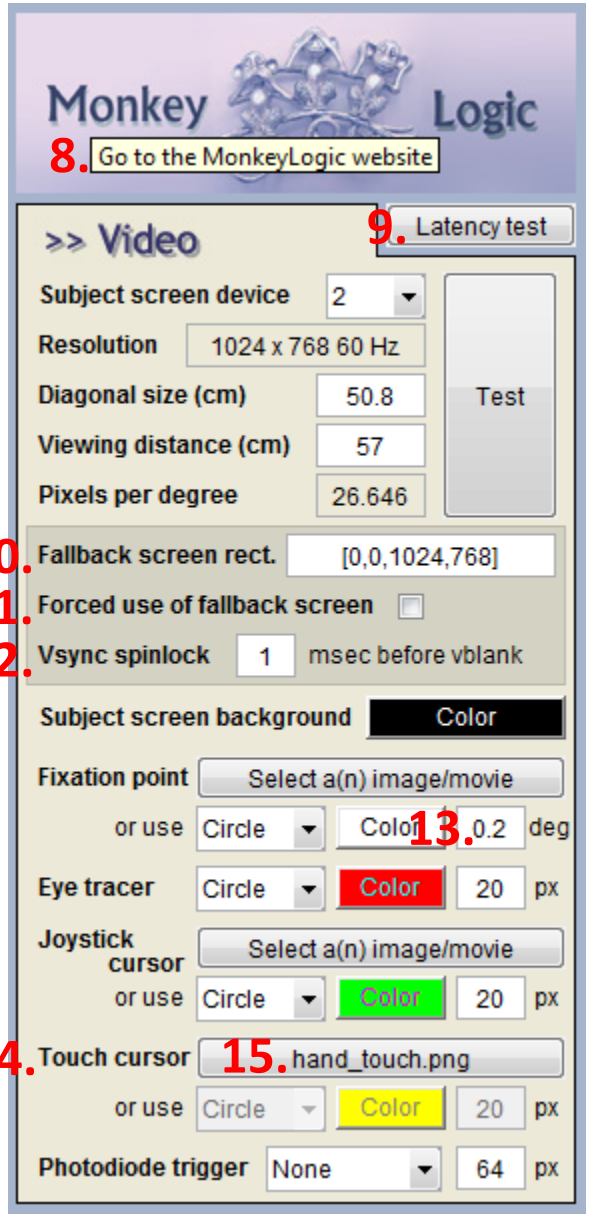
- This is useful to create the subject screen across multiple monitors. There can be some conditions and limitations.

12. The wait time for screen flips is adjustable.

13. The circle sizes are all in radius in NIMH ML 2.

14. Touchscreen tracer

15. Transparent image by alpha blending (PNG format)



User Interface (3/4)

16. Error-free DAQ setup

- Boards and subsystems that do not support the selected signal type and channels/ports and lines assigned already are hidden from the UI.

17. Touchscreen support

18. USB/Serial joystick support

19. AI Sample rate affects only the data saved to the file.

- The internal sampling rate is always 1 kHz for behavior monitoring.

20. Online smoothing of analog input signals

21. Reward function customization

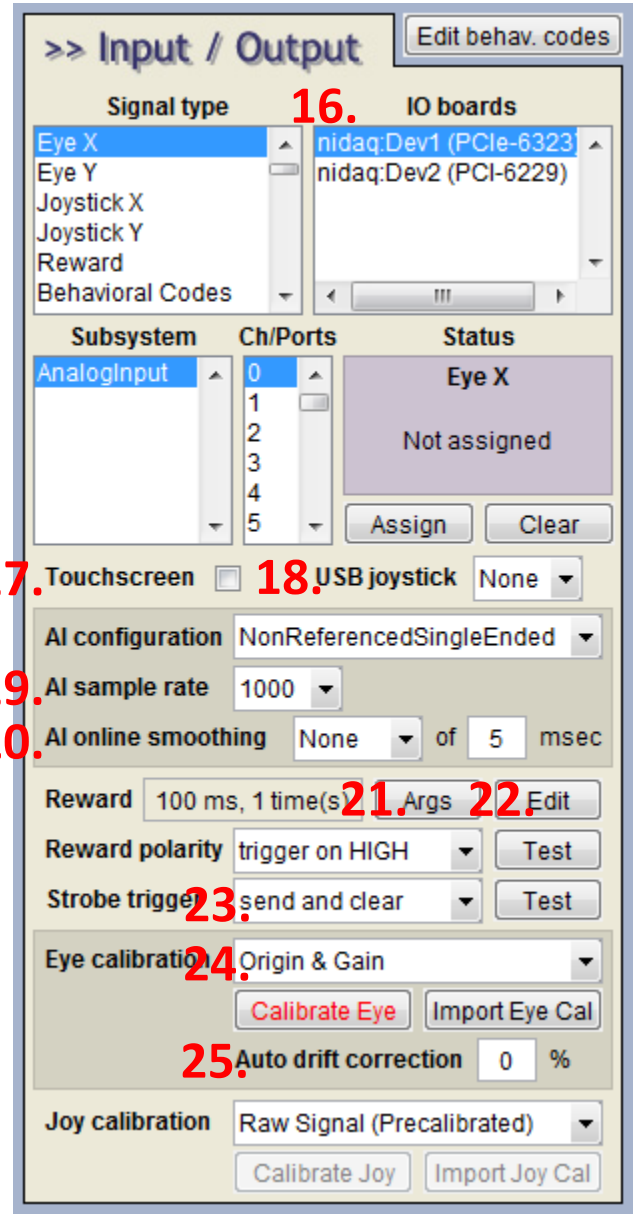
- Now multichannel reward devices can be triggered via goodmonkey().

22. Edit button for the custom reward function

23. New strobe option ("send and clear") that does not require Strobe Bit.

24. New calibration method option ("Origin & Gain")

25. New drift correction method that does not need signal smoothing



User Interface (4/4)

26. Button that opens the online manual for error handling, condition selection and block selection

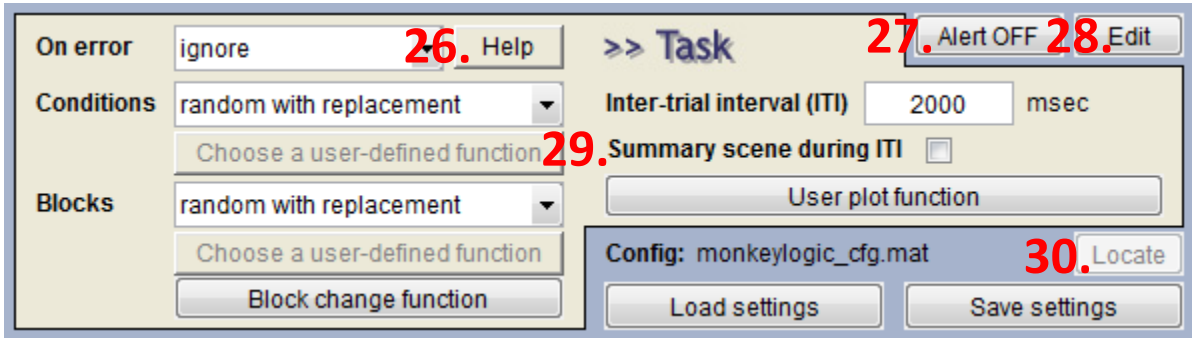
27. Alert ON/OFF

- When enabled, a customizable alert function (alert_function.m) is called when the task starts/ends, when a new block starts/ends and when a new trial starts/ends so that the function can notify users of the progress of the experiment.

28. Edit button for the custom alert function

29. Summary screen during the inter-trial interval can be turned off.

30. Button that opens the configuration file folder (usually the task file directory).



Changes of the Conditions File Syntax in NIMH ML 2

- **TaskObjects**

  - You can specify a transparent color for the PIC objects when the image does not contain the alpha channel data.

    pic(filename, Xpos, Ypos[, colorkey]) % [r g b]
    pic(filename, Xpos, Ypos, Width, Height[, colorkey])

  - The STM object can be re-armed immediately after stimulation, for multiple triggering.

    stm(port, datasource[, retriggering]) % 0 or 1

- **Subdirectory name & file extension of the stimulus file/data source**

  - You can add the subdirectory name and the file extension to the filename or the data source name in the conditions file.

Changes of the Timing File Syntax in NIMH ML 2 (1/3)

- **Changes in trial-specific functions**

  - trialtime() does not return the frame number any more.

```
[t, framenumber] = trialtime;  % not correct any more in NIMH ML 2
t = trialtime;  % correct
```

Changes of the Timing File Syntax in NIMH ML 2 (2/3)

- **New trial-specific functions**

- mouse_position() returns the current position of the mouse and its button status.

```
[xy button] = mouse_position();
```

- rewind_movie() moves the playback position of the given movie.

```
rewind_movie(taskobject_no, time_in_msec);
```

- get_movie_duration() returns the length of the selected movie.

```
duration_in_msec = get_movie_duration(taskobject_no);
```

- dashboard() displays 3 lines of user strings on the top of the control screen.
In NIMH MonkeyLogic 2, user_text() and user_warning() are updated only at the end of a trial. However, dashboard() is updated immediately as soon as the new string is set.

```
dashboard(line_no, text, text_color);  % line_no: 1-3
```

Changes of the Timing File Syntax in NIMH ML 2 (3/3)

- **The TrialRecord Structure**

- TrialRecord is now a class object, not a struct, so it is not allowed to add new fields to it.

→ Move user-defined fields under the "User" field.

```
TrialRecord.var1 = 200;  % fine in the old ML, but not in NIMH ML 2
TrialRecord.User.var1 = 200;  % good with NIMH ML 2
```

- TrialRecord has 3 additional fields that users can set freely in the timing code.

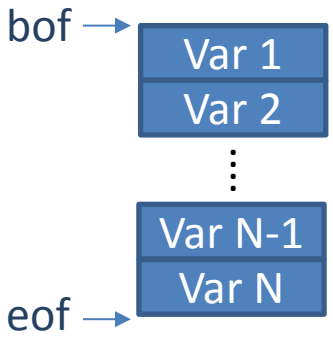
TrialRecord.BlockChange: When set to be true (or 1), a new block begins in the next trial.

TrialRecord.Pause: When set to be true (or 1), the task stops and the pause menu is displayed after the current trial ends.

TrialRecord.Quit: When set to be true (or 1), the task exits at the end of the current trial without showing the pause menu.

BHV2 Format (1/4)

- BHV2 has no file header and just contains the contents of variables.



- Each variable block starts with 6 fields like the following.

Field	Type	Length
Length of Variable Name (LN)	uint64	1
Variable Name	char*1	LN
Length of Variable Type (LT)	uint64	1
Variable Type	char*1	LT
Dimension of Variable (DV)	uint64	1
Size of Variable	uint64	DV

The 1st, 3rd, 5th fields Indicates the lengths of the 2nd, 4th, 6th fields, respectively.

BHV2 Format (2/4)

- If the variable type is one of the MATLAB primitive data types (char, integers, single, double, logical), then the content of the variable follows those 6 fields in **column-major order**. For example, if A = rand(2,2), the byte order of A will be like this.

1	A	6	double	2	[2 2]	A(1,1)	A(2,1)	A(1,2)	A(2,2)

- If the variable type is **struct**, there is one more field of uint64 that indicates the number of fields. Then the first field of the first struct array starts.
  ex) A = [struct('a',1,'b','xyz') struct('a',9,'b','')] % the number of fields is 2; a & b.

1	A	6	struct	2	[1 2]	2	A(1).a	A(1).b	A(2).a
A(2).b									

- If the variable type is **cell**, the cells of the cell array comes in column-major order.
  ex) A = cell(3,2)

1	A	4	cell	2	[3 2]	A{1,1}	A{2,1}	A{3,1}	A{1,2}
A{2,2}	A{3,2}								

BHV2 Format (3/4)

- Example of the byte order of a struct

```
1           [1x1 uint64]    % length('A')
A           [1x1 char]      % struct name
6           [1x1 uint64]    % length('struct')
struct      [1x6 char]      % variable type
2           [1x1 uint64]    % dimension of variable
[1 2]       [1x2 double]    % size of struct
2           [1x1 uint64]    % number of fields in A
1           [1x1 uint64]    % length('a')
a           [1x1 char]      % name of the first field
6           [1x1 uint64]    % length('double')
double      [1x6 char]      % variable type
2           [1x1 uint64]    % dimension of variable
[1 3]       [1x2 double]    % size of variable
1 2 3       [1x3 double]    % content of the first field
1           [1x1 uint64]    % length('b')
b           [1x1 char]      % name of the second field
4           [1x1 uint64]    % length('char')
char        [1x4 char]      % variable type
2           [1x1 uint64]    % dimension of variable
[1 3]       [1x3 double]    % size of variable
xyz         [1x3 char]      % content of the second field
1           [1x1 uint64]    % length('a')
a           [1x1 char]      % name of the first field
6           [1x1 uint64]    % length('double')
double      [1x6 char]      % variable type
2           [1x1 uint64]    % dimension of variable
[2 2]       [1x2 double]    % size of variable
5 7 6 8     [2x2 double]    % content of the first field
```

(continue on the right column)

```
A(1).a = [1 2 3];
A(1).b = 'xyz';

A(2).a = [5 6; 7 8];
A(2).b = '';
```

```
1           [1x1 uint64]    % length('b')
b           [1x1 char]      % name of the second field
4           [1x1 uint64]    % length('char')
char        [1x4 char]      % variable type
2           [1x1 uint64]    % dimension of variable
[0 0]       [1x2 double]    % size of variable
''          [0x0 char]      % content of the second field
(end)
```

The last byte in the above example does not exist since its content is blank. Note that the content of A(2).a is written like [5 7 6 8], not [5 6 7 8], since arrays are in column major order in MATLAB.

BHV2 Format (4/4)

- Example of the byte order of a cell

```
1           [1x1 uint64]    % length('A')
A           [1x1 char]      % cell array name
4           [1x1 uint64]    % length('cell')
cell        [1x4 char]      % variable type
2           [1x1 uint64]    % dimension of variable
[2 2]       [1x2 double]    % size of cell array
0           [1x1 uint64]    % A{1,1} doesn't have name
''          [0x0 char]      % no name
6           [1x1 uint64]    % length('double')
double      [1x6 char]      % variable type
2           [1x1 uint64]    % dimension of variable
[1 3]       [1x2 double]    % size of variable
1 2 3       [1x3 double]    % content of the A{1,1}
0           [1x1 uint64]    % A{2,1} doesn't have name
''          [0x0 char]      % no name
6           [1x1 uint64]    % length('double')
double      [1x6 char]      % variable type
2           [1x1 uint64]    % dimension of variable
[2 2]       [1x2 double]    % size of variable
5 7 6 8     [2x2 double]    % content of A{2,1}
0           [1x1 uint64]    % A{1,2} has no name
''          [0x0 char]      % no name
4           [1x1 uint64]    % length('char')
char        [1x4 char]      % variable type
2           [1x1 uint64]    % dimension of variable
[1 3]       [1x2 double]    % size of variable
xyz         [1x3 double]    % content of A{1,2}
```

(continue on the right column)

```
A = cell(2,2);
A{1,1} = [1 2 3];
A{1,2} = 'xyz';
A{2,1} = [5 6; 7 8];
A{2,2} = '';
```

```
0           [1x1 uint64]    % A{2,2} has no name
''          [0x0 char]      % no name
4           [1x1 uint64]    % length('char')
char        [1x4 char]      % variable type
2           [1x1 uint64]    % dimension of variable
[0 0]       [1x2 double]    % size of variable
''          [0x0 char]      % content of A{2,2}
(end)
```

Again the fields that have any 0-sized dimension are not written to the file. And not only a double matrix (A{2,1}) but also a cell array ('A' itself) is arranged in column major order.

H5 Format

- Creating and reading .h5 files are implemented with MATLAB's low-level HDF5 access functions. In BHV2, the name, type and size of each variable are stored in the 6 fields that come at the beginning of each variable block. In H5, those fields are stored as attributes of datasets or groups.

- The HDF5 implementation in MATLAB R2014b or earlier does not allow to create a 0-sized dataspace. So the h5 files created in those versions contains a single '0', even when the variable is empty. When you read those datasets in your application, you should discard '0' when the size of the variable written in the attribute value is 0.

- The MATLAB primitive data types are stored in datasets. The struct and cell arrays are created as groups of the fields or cells.

- H5 can be read with any common HDF5 tool, like HDFView (https://support.hdfgroup.org/products/java/hdfview/).

- Refer to mlhdf5.m for implementation details.

New runtime functions

- Two new functions that replace toggleobject() & eyejoytrack()
  - scene = create_scene(adapter, stimuli);
  - fliptime = run_scene(scene, eventcode);
- The “adapter” is a MATLAB class object and a building block of a scene.
  - The adapters can be concatenated to handle complex behavior analysis and graphics.
  - The adapter has two member functions, analyze() and draw(), which are called by run_scene() every frame.
  - analyze() sets two status variables, Success and continue_, to indicate whether the target behavior is detected and whether the next frame will be continued.