Experiment runner

Release 1.3.1

MIT

Nov 07, 2020

STUDY PROTOCOL

Building a study on Lookit and using Lookit's own experiment runner? You're in the right place to learn how to put together the study protocol you have in mind. To the left you can find information about. . .

ONE

SPECIFYING YOUR STUDY PROTOCOL

There are some features that are common to all frames, like being able to set a `generateProperties` function to adjust parameters on-the-fly based on what the child has done so far:

1.1 Protocol configuration

Researchers specify how their Lookit study works by writing a "protocol configuration" for their study. This configuration is written in JSON, which stands for JavaScript Object Notation - this is just a special text format, not code.

Your study protocol configuration tells the Lookit experiment runner what sequence of "frames" to use in your study, and set all the options for those frames like what pictures or videos to show and for how long.

1.1.1 JSON format

No programming is required to design a study: JSON is a simple, human-readable text format for describing data. A JSON object is an unordered set of key – value pairs, with the following rules:

- The object itself is enclosed in curly braces.

- Keys are unique strings enclosed in double quotes.

- A key and value are separated by a colon.

- Key-value pairs are separated by commas.

A JSON value can be any of the following:

- a string (enclosed in double quotes)

- a number

- a JSON object (as described above)

- an array (an ordered list of JSON values, separated by commas and enclosed by square brackets)

- true

- false

- null

There are no requirements for specific formatting of a JSON document (whitespace that isn't part of a string is ignored). Here is an example JSON object to illustrate these principles:

```
{
    "name": "Jane",
    "age": 43,
    "favoritefoods": [
        "eggplant",
        "apple",
        "lima beans"
    ],
    "allergies": {
        "peanut": "mild",
        "shellfish": "severe"
    }
}
```

The keys are the strings `name`, `age`, `favoritefoods`, and `allergies`. Favorite foods are stored as an array, or ordered list; allergies are stored as a JSON object mapping food names to severity of reaction. The same object could also be written as follows, in a different order and with none of the formatting: `{"age": 43, "allergies": {"peanut": "mild", "shellfish": "severe"}, "name": "Jane", "favoritefoods": ["eggplant", "apple", lima beans"]}`

A helpful resource to check your JSON Schema for simple errors like missing or extra commas, unmatched braces, etc. is jsonlint.

The JSON you write for your protocol configuration gets interpreted by Lookit's experiment runner, which expects to find specific types of information in the configuration file. (Formally, it expects the data to conform to a custom JSON schema.)

1.1.2 Study protocol structure

Studies on Lookit are broken into a set of fundamental units called **frames**, which can also be thought of as "pages" of the study. A single experimental trial (e.g. looking time measurement) would generally be one frame, as are the video consent procedure and exit survey. Your JSON must have two keys: `frames` and `sequence`. The `frames` value defines the frames used in this study: it must be a JSON object mapping frame nicknames (any unique strings chosen by the researcher) to frame objects (defined next). The `sequence` value must be an ordered list of the frames to use in this study; values in this list must be frame nicknames from the "frames" value.

Here is the JSON for a very minimal Lookit study:

```
{
    "frames": {
        "my-consent-frame": {
            "kind": "exp-video-consent",
            "prompt": "I agree to participate",
            "blocks": [
                {
                    "title": "About the study",
                    "text": "This isn't a real study."
                }
            ]
        },
        "my-exit-survey": {
            "kind": "exp-lookit-exit-survey",
            "debriefing": {
                "title": "Thank you!",
                "text": "You participated."
            }
```

```
            }
        },
        "sequence": [
            "my-consent-frame",
            "my-exit-survey"
        ]
}
```

This JSON specifies a Lookit study with two frames, consent and an exit survey. Note that the frame nicknames `my-consent-frame` and `my-exit-survey` that are defined in `frames` are also used in the `sequence`. Frames may be specified but not used in `sequence`. Here's the object associated with the `my-exit-survey` frame:

```
{
    "kind": "exp-lookit-exit-survey",
    "debriefing": {
        "title": "Thank you!",
        "text": "You participated."
    }
}
```

Within each frame object, a `kind` must be specified. This determines the frame type that will be used. Additional data may be included in the frame object to customize the behavior of the frame, for instance to specify instruction text or the stimuli to use for a test trial. The keys that may (or must) be included in a frame object are determined by the frame type; each frame definition includes a JSON Schema describing the expected data to be passed. Multiple frames of the same kind may be included in a study – for instance, test trials using different stimuli.

The separation of frame definitions and sequence allows researchers to easily and flexibly edit and test study protocols – for instance, the order of frames may be altered or a particular frame removed for testing purposes without altering any frame definitions.

1.2 Protocol generators

Although your study protocol JSON can be configured to handle a wide range of common condition assignment, counterbalancing, and conditional logic schemes, in some cases it may still be more natural to programmatically generate the protocol. For instance, you might want to use a protocol generator in cases of:

- Complex counterbalancing schemes: for instance, ensuring you use each image from each set exactly once for training and once for test in a preferential looking task, without repeating any pairings

- Complex stimulus/protocol generation: for instance, you want to generate a random Sudoku board and create a sequence of frames guiding the child through it

- Complex longitudinal assignment: for instance, you have three tasks to be completed between 6 and 12 months and three tasks to be completed between 18 and 24 months, and want to assign children to the appropriate task based on their age and what they've already completed

1.2.1 What `generateProtocol` should return

Your protocol generator should return a single object with fields "frames" and "sequence" - exactly like the protocol specification JSON. In fact, JSON is valid Javascript: you can copy and paste your protocol specification right into the generator function to get started!

1.2.2 What information it has access to

The `generateProtocol` function receives two arguments, `child` and `pastSessions`.

The `child` and the elements of the array `pastSessions` are Ember objects and do not behave exactly like regular Javascript objects. Here

`child` is an Ember object representing the child currently participating in this study. You can access the following fields using `child.get(<fieldName>)`, e.g. `child.get('givenName')`:

- givenName (string)

- birthday (Date)

- gender (string, 'm' / 'f' / 'o')

- ageAtBirth (string, e.g. '25 weeks'. One of '40 or more weeks', '39 weeks' through '24 weeks', 'Under 24 weeks', or 'Not sure or prefer not to answer')

- additionalInformation (string) freeform "anything else we should know" field on child registration

- languageList (string) space-separated list of languages child is exposed to (2-letter codes)

- conditionList (string) space-separated list of conditions/characteristics of child from registration form, as used in criteria expression - e.g. "autism_spectrum_disorder deaf multiple_birth"

`pastSessions` is an array of Ember objects representing past sessions for this child and this study, in reverse time order: pastSessions[0] is THIS session, pastSessions[1] the previous sessions, and so on. Each session has the following fields, corresponding to information available for download on Lookit, which can be accessed as `pastSessions[i].get(<fieldName>)`:

- createdOn (Date)

- conditions

- expData

- sequence

- completed

- globalEventTimings

- completedConsentFrame (note - this list will include even "responses") where the user did not complete the consent form!

- demographicSnapshot

- isPreview

1.2.3 Empty template

Here is the default value for the protocol generator - an empty template with comments explaining what the arguments are.

```javascript
function generateProtocol(child, pastSessions) {
    /*
     * Generate the protocol for this study.
     *
     * @param {Object} child
     *    The child currently participating in this study. Includes fields:
     *        givenName (string)
     *        birthday (Date)
     *        gender (string, 'm' / 'f' / 'o')
     *        ageAtBirth (string, e.g. '25 weeks'. One of '40 or more weeks',
     *            '39 weeks' through '24 weeks', 'Under 24 weeks', or
     *            'Not sure or prefer not to answer')
     *        additionalInformation (string)
     *        languageList (string) space-separated list of languages child is
     *            exposed to (2-letter codes)
     *        conditionList (string) space-separated list of conditions/characteristics
     *            of child from registration form, as used in criteria expression
     *            - e.g. "autism_spectrum_disorder deaf multiple_birth"
     *
     *        Use child.get to access these fields: e.g., child.get('givenName') returns
     *        the child's given name.
     *
     * @param {!Array<Object>} pastSessions
     *        List of past sessions for this child and this study, in reverse time order:
     *        pastSessions[0] is THIS session, pastSessions[1] the previous session,
     *        back to pastSessions[pastSessions.length - 1] which has the very first
     *        session.
     *
     *        Each session has the following fields, corresponding to values available
     *        in Lookit:
     *
     *        createdOn (Date)
     *        conditions
     *        expData
     *        sequence
     *        completed
     *        globalEventTimings
     *        completedConsentFrame (note - this list will include even "responses")
     *            where the user did not complete the consent form!
     *        demographicSnapshot
     *        isPreview
     *
     * @return {Object} Protocol specification for Lookit study; object with 'frames'
     *    and 'sequence' keys.
     */

    // Return a study protocol with "frames" and "sequence" fields just like when
    // defining the protocol in JSON only
    return {
        frames: {},
        sequence: []
    };
}
```

1.2.4 Examples

Returning different protocols based on age

Here is a simple "study" where the protocol returned is different depending on the child's age:

```
function generateProtocol(child, pastSessions) {

    let one_day = 1000 * 60 * 60 * 24; // ms in one day
    let child_age_in_days = -1;
    try {
        child_age_in_days = (new Date() - child.get('birthday')) / one_day;
    } catch (error) {
        // Display what the error was for debugging, but continue with fake
        // age in case we can't calculate age for some reason
        console.error(error);
    }
    child_age_in_days = child_age_in_days || -1; // If undefined/null, set to default

    // Define frames that will be used for both the baby and toddler versions of the
→study
    let frames = {
        "video-config": {
            "kind": "exp-video-config",
            "troubleshootingIntro": "If you're having any trouble getting your webcam
→set up, please feel free to email the XYZ lab at xyz@abc.edu and we'd be glad to
→help out!"
        },
        "video-consent": {
            "kind": "exp-lookit-video-consent",
            "PIName": "Jane Smith",
            "datause": "We are primarily interested in your child's emotional
→reactions to the images and sounds. A research assistant will watch your video to
→measure the precise amount of delight in your child's face as he or she sees each
→cat picture.",
            "payment": "After you finish the study, we will email you a $5 BabyStore
→gift card within approximately three days. To be eligible for the gift card your
→child must be in the age range for this study, you need to submit a valid consent
→statement, and we need to see that there is a child with you. But we will send a
→gift card even if you do not finish the whole study or we are not able to use your
→child's data! There are no other direct benefits to you or your child from
→participating, but we hope you will enjoy the experience.",
            "purpose": "Why do babies love cats? This study will help us find out
→whether babies love cats because of their soft fur or their twitchy tails.",
            "PIContact": "Jane Smith at 123 456 7890",
            "procedures": "Your child will be shown pictures of lots of different
→cats, along with noises that cats make like meowing and purring. We are interested
→in which pictures and sounds make your child smile. We will ask you (the parent) to
→turn around to avoid influencing your child's responses. There are no anticipated
→risks associated with participating.",
            "institution": "Science University"
        },
        "exit-survey": {
            "kind": "exp-lookit-exit-survey",
            "debriefing": {
                "text": "Here is where you would enter debriefing information for the
→family. This is a chance to explain the purpose of your study and how the family
→helped. At this point it's more obvious to the participant that skimming the info
→is fine if they're not super-interested, so you can elaborate in ways you might
→have avoided ahead of time in the interest of keeping instructions short. You may
→want to mention the various conditions kids were assigned to if you didn't before,
→and try to head off any concerns parents might have about how their child did on
→the study, especially if there are 'correct' answers that will have been obvious to
→a parent. <br><br> It is great if you can link people to a layperson-accessible
→article on a related topic - e.g., media coverage of one of your previous studies
```

Chapter 1. Specifying your study protocol

```
                "title": "Thank you!"
            }
        }
    }

    // Add a "test frame" that's different depending on the child's age.
    // You could actually be defining whole separate protocols here (e.g. for
    // a longitudinal study with a bunch of timepoints), using different stimuli
    // in the same frames, just customizing instructions, etc.

    // If the age is -1 because there was some error, they'll get the baby version.
    if (child_age_in_days <= 365) {
        frames["test-frame"] = {
            "kind": "exp-lookit-instructions",
            "blocks": [
                {
                    "title": "[Example text for BABY version of study]",
                    "listblocks": [
                        {
                            "text": "Lorem ipsum dolor sit amet, consectetur␣
↪adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.␣
↪Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip␣
↪ex ea commodo consequat."
                        },
                        {
                            "text": "Duis aute irure dolor in reprehenderit in␣
↪voluptate velit esse cillum dolore eu fugiat nulla pariatur."
                        }
                    ]
                }
            ],
            "showWebcam": false,
            "nextButtonText": "Finish up"
        };
    } else {
        frames["test-frame"] = {
            "kind": "exp-lookit-instructions",
            "blocks": [
                {
                    "title": "[Example text for TODDLER version of study]",
                    "listblocks": [
                        {
                            "text": "Lorem ipsum dolor sit amet, consectetur␣
↪adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.␣
↪Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip␣
↪ex ea commodo consequat."
                        },
                        {
                            "text": "Duis aute irure dolor in reprehenderit in␣
↪voluptate velit esse cillum dolore eu fugiat nulla pariatur."
                        }
                    ]
                }
            ],
            "showWebcam": false,
            "nextButtonText": "Finish up"
        }
```

```
    }

    // Sequence of frames is the same in both cases, the 'test-frame' will just
    // be differently defined base on age.
    let frame_sequence = ['video-config', 'video-consent', 'test-frame', 'exit-survey
↪']

    // Return a study protocol with "frames" and "sequence" fields just like when
    // defining the protocol in JSON only
    return {
        frames: frames,
        sequence: frame_sequence
    };
}
```

Alternating question types each session

Here is an example of using information in pastSessions to determine the protocol for this session. In this case, we just look at what the child did last, and switch to the opposite trial type for this time. (Note that this could also be done using the conditionForAdditionalSessions parameter of the random-parameter-set randomizer; however, more complex longitudinal designs may benefit from programmatic specification.)

```
function generateProtocol(child, pastSessions) {

    // Assign condition randomly as fallback/initial value. This will be true/false
    // with equal probability.
    let is_happy_condition = Math.random() > 0.5;

    try {
        // First, find the most recent session where the participant got to the point
        // of the "test trial"
        var mostRecentSession = pastSessions.find(
            sess => Object.keys(sess.get('expData', {})).some(frId => frId.endsWith('-
↪match-emotion')));
        // If there is such a session, find out what condition they were in that time
        // and flip it
        if (mostRecentSession) {
            let expData = mostRecentSession.get('expData', {});
            let frameKey = Object.keys(expData).find(frId => frId.endsWith('-match-
↪emotion'));
            // Flip condition from last time: do happy condition this time if last
            // time 'happy' was NOT in the *-match-emotion frame ID
            is_happy_condition = !(frameKey.includes('happy'));
        }
    } catch (error) {
        // Just in case - wrap the above in a try block so we fall back to
        // random assignment if something is weird about the pastSessions data
        console.error(error);
    }


    // Define all possible frames that might be used
    let frames = {
        "intro": {
            "blocks": [{
```

```
                "text": "Sed ut perspiciatis unde omnis iste natus error sit␣
→voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab␣
→illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo.",
                "title": "[Introduction frame]"
            },
            {
                "text": "Nemo enim ipsam voluptatem quia voluptas sit aspernatur␣
→aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem␣
→sequi nesciunt."
            },
            {
                "text": "Neque porro quisquam est, qui dolorem ipsum quia dolor␣
→sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora␣
→incidunt ut labore et dolore magnam aliquam quaerat voluptatem."
            }
        ],
        "showPreviousButton": false,
        "kind": "exp-lookit-text"
    },
    "happy-match-emotion": {
        "kind": "exp-lookit-images-audio",
        "audio": "matchremy",
        "images": [{
                "id": "cue",
                "src": "happy_remy.jpg",
                "position": "center",
                "nonChoiceOption": true
            },
            {
                "id": "option1",
                "src": "happy_zenna.jpg",
                "position": "left",
                "displayDelayMs": 2000
            },
            {
                "id": "option2",
                "src": "annoyed_zenna.jpg",
                "position": "right",
                "displayDelayMs": 2000
            }
        ],
        "baseDir": "https://www.mit.edu/~kimscott/placeholderstimuli/",
        "autoProceed": false,
        "doRecording": false,
        "choiceRequired": true,
        "parentTextBlock": {
            "text": "Some explanatory text for parents",
            "title": "For parents"
        },
        "canMakeChoiceBeforeAudioFinished": true
    },
    "sad-match-emotion": {
        "kind": "exp-lookit-images-audio",
        "audio": "matchzenna",
        "images": [{
                "id": "cue",
                "src": "sad_zenna.jpg",
```

```
                        "position": "center",
                        "nonChoiceOption": true
                    },
                    {
                        "id": "option1",
                        "src": "surprised_remy.jpg",
                        "position": "left",
                        "feedbackAudio": "negativefeedback",
                        "displayDelayMs": 3500
                    },
                    {
                        "id": "option2",
                        "src": "sad_remy.jpg",
                        "correct": true,
                        "position": "right",
                        "displayDelayMs": 3500
                    }
                ],
                "baseDir": "https://www.mit.edu/~kimscott/placeholderstimuli/",
                "autoProceed": false,
                "doRecording": false,
                "choiceRequired": true,
                "parentTextBlock": {
                    "text": "Some explanatory text for parents",
                    "title": "For parents"
                },
                "canMakeChoiceBeforeAudioFinished": true
            },
            "exit-survey": {
                "kind": "exp-lookit-exit-survey",
                "debriefing": {
                    "text": "At vero eos et accusamus et iusto odio dignissimos ducimus
→qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas
→molestias excepturi sint occaecati cupiditate non provident, similique sunt in
→culpa qui officia deserunt mollitia animi, id est laborum et dolorum fuga. Et harum
→quidem rerum facilis est et expedita distinctio. <br> <br> Nam libero tempore, cum
→soluta nobis est eligendi optio cumque nihil impedit quo minus id quod maxime
→placeat facere possimus, omnis voluptas assumenda est, omnis dolor repellendus.
→Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe
→eveniet ut et voluptates repudiandae sint et molestiae non recusandae. <br> <br>
→Itaque earum rerum hic tenetur a sapiente delectus, ut aut reiciendis voluptatibus
→maiores alias consequatur aut perferendis doloribus asperiores repellat.",
                    "title": "Thank you!"
                }
            }
        }
    }

    // Construct the sequence based on the condition.
    let frame_sequence = [
        'intro',
        is_happy_condition ? "happy-match-emotion" : "sad-match-emotion",
        'exit-survey'
    ]

    // Return a study protocol with "frames" and "sequence" fields just like when
    // defining the protocol in JSON only
    return {
```

```
        frames: frames,
        sequence: frame_sequence
    };
}
```

Randomizing but preventing re-use of stimuli across trials

Here is an example of a counterbalancing structure that benefits from being able to programmatically describe the study protocol.

On each test trial, the infant sees two images, one on the right and one on the left, from two different categories. Audio matching one of the two categories is played. There are three categories: "adorable," "delicious," and "exciting." The infant should see each possible pairing of categories twice, once with audio matching each category. This makes six trials (adorable-delicious, adorable-exciting, delicious-exciting x 2 audio choices each). There are four images for each category. Each should be used exactly once during the study. The left/right placement of the images should be determined randomly.

```javascript
function generateProtocol(child, pastSessions) {

    // -------- Helper functions -------------------------------------------

    // See http://stackoverflow.com/a/12646864
    // Returns a new array with elements of the array in random order.
    function shuffle(array) {
        var shuffled = Ember.$.extend(true, [], array); // deep copy array
        for (var i = array.length - 1; i > 0; i--) {
            var j = Math.floor(Math.random() * (i + 1));
            var temp = shuffled[i];
            shuffled[i] = shuffled[j];
            shuffled[j] = temp;
        }
        return shuffled;
    }

    // Returns a random element of an array, and removes that element from the array
    function pop_random(array) {
        var randIndex = Math.floor(Math.random() * array.length);
        if (array.length) {
            return array.pop(randIndex);
        }
        return null
    }

    // -------- End helper functions ---------------------------------------

    // Define common (non-test-trial) frames
    let frames = {
        "video-config": {
            "kind": "exp-video-config",
            "troubleshootingIntro": "If you're having any trouble getting your webcam␣
→set up, please feel free to email the XYZ lab at xyz@abc.edu and we'd be glad to␣
→help out!"
        },
        "video-consent": {
            "kind": "exp-lookit-video-consent",
```

```
            "PIName": "Jane Smith",
            "datause": "We are primarily interested in your child's emotional␣
↪reactions to the images and sounds. A research assistant will watch your video to␣
↪measure the precise amount of delight in your child's face as he or she sees each␣
↪cat picture.",
            "payment": "After you finish the study, we will email you a $5 BabyStore␣
↪gift card within approximately three days. To be eligible for the gift card your␣
↪child must be in the age range for this study, you need to submit a valid consent␣
↪statement, and we need to see that there is a child with you. But we will send a␣
↪gift card even if you do not finish the whole study or we are not able to use your␣
↪child's data! There are no other direct benefits to you or your child from␣
↪participating, but we hope you will enjoy the experience.",
            "purpose": "Why do babies love cats? This study will help us find out␣
↪whether babies love cats because of their soft fur or their twitchy tails.",
            "PIContact": "Jane Smith at 123 456 7890",
            "procedures": "Your child will be shown pictures of lots of different␣
↪cats, along with noises that cats make like meowing and purring. We are interested␣
↪in which pictures and sounds make your child smile. We will ask you (the parent) to␣
↪turn around to avoid influencing your child's responses. There are no anticipated␣
↪risks associated with participating.",
            "institution": "Science University"
        },
        "exit-survey": {
            "kind": "exp-lookit-exit-survey",
            "debriefing": {
                "text": "At vero eos et accusamus et iusto odio dignissimos ducimus␣
↪qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas␣
↪molestias excepturi sint occaecati cupiditate non provident, similique sunt in␣
↪culpa qui officia deserunt mollitia animi, id est laborum et dolorum fuga. Et harum␣
↪quidem rerum facilis est et expedita distinctio. <br> <br> Nam libero tempore, cum␣
↪soluta nobis est eligendi optio cumque nihil impedit quo minus id quod maxime␣
↪placeat facere possimus, omnis voluptas assumenda est, omnis dolor repellendus.␣
↪Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe␣
↪eveniet ut et voluptates repudiandae sint et molestiae non recusandae. <br> <br>␣
↪Itaque earum rerum hic tenetur a sapiente delectus, ut aut reiciendis voluptatibus␣
↪maiores alias consequatur aut perferendis doloribus asperiores repellat.",
                "title": "Thank you!"
            }
        }
    }

    // Start off the frame sequence with config/consent frames; we'll add test
    // trials as we construct them
    let frame_sequence = ['video-config', 'video-consent']

    // start at a random point in this list and cycle through across trials.
    // each element is a list: category1, category2, audio.
    // category1 and category2 match up to keys in available_images; audio
    // should be filenames in baseDir/mp3
    let all_category_pairings = [
        [
            "adorable",
            "delicious",
            "Adorable"
        ],
        [
            "adorable",
```

```
            "delicious",
            "Delicious"
        ],
        [
            "delicious",
            "exciting",
            "Delicious"
        ],
        [
            "delicious",
            "exciting",
            "Exciting"
        ],
        [
            "adorable",
            "exciting",
            "Adorable"
        ],
        [
            "adorable",
            "exciting",
            "Exciting"
        ]
    ]

    // Every image is just used once total, either as a target or as a distractor.
    // We'll remove the images from these lists as they get used.
    let available_images = {
        "adorable": [
            "Adorable_1.png",
            "Adorable_2.png",
            "Adorable_3.png",
            "Adorable_4.png"
        ],
        "delicious": [
            "Delicious_1.png",
            "Delicious_2.png",
            "Delicious_3.png",
            "Delicious_4.png"
        ],
        "exciting": [
            "Exciting_1.png",
            "Exciting_2.png",
            "Exciting_3.png",
            "Exciting_4.png"
        ]
    }

    // Make a deep copy of the original available images, in case we run out
    // (e.g. after adding additional trials) and need to "refill" a category.
    let all_images = Ember.$.extend(true, {}, available_images)

    // Choose a random starting point and order for the category pairings
    let ordered_category_pairings = shuffle(all_category_pairings)

    for (iTrial = 0; iTrial < 6; iTrial++) {
```

```javascript
        let category_pairing = ordered_category_pairings[iTrial]
        let category_id_1 = category_pairing[0]
        let category_id_2 = category_pairing[1]
        let audio = category_pairing[2]

        // "Refill" available images if empty
        if (!available_images[category_id_1].length) {
            available_images[category_id_1] = all_images[category_id_1]
        }
        if (!available_images[category_id_2].length) {
            available_images[category_id_2] = all_images[category_id_2]
        }

        let image1 = pop_random(available_images[category_id_1])
        let image2 = pop_random(available_images[category_id_2])

        let left_right_pairing = shuffle(["left", "right"])

        thisTrial = {
            "kind": "exp-lookit-images-audio",
            "audio": audio,
            "images": [{
                    "id": "option1-test",
                    "src": image1,
                    "position": left_right_pairing[0]
                },
                {
                    "id": "option2-test",
                    "src": image2,
                    "position": left_right_pairing[1]
                }
            ],
            "baseDir": "https://raw.githubusercontent.com/schang198/lookit-stimuli-
↪template/master/",
            "pageColor": "gray",
            "audioTypes": [
                "mp3"
            ],
            "autoProceed": true
        }

        // Store this frame in frames and in the sequence
        frameId = 'test-trial-' + (iTrial + 1)
        frames[frameId] = thisTrial;
        frame_sequence.push(frameId);
    }

    // Finish up the frame sequence with the exit survey
    frame_sequence = frame_sequence.concat(['exit-survey'])

    // Return a study protocol with "frames" and "sequence" fields just like when
    // defining the protocol in JSON only
    return {
        frames: frames,
        sequence: frame_sequence
    };
}
```

Customizing text based on the child's gender

Here is a snippet showing how you might generate text to use in a study to match the child's gender:

```
let gender = child.get('gender');
let _KID = 'kid';
let _THEY = 'they';
let _THEIR = 'their';
let _THEIRS = 'theirs';
let _THEM = 'them';

if (gender == 'f') {
    _KID = 'girl';
    _THEY = 'she';
    _THEIR = 'her';
    _THEIRS = 'hers';
    _THEM = 'her';
} else if (gender == 'm') {
    _KID = 'boy';
    _THEY = 'he';
    _THEIR = 'his';
    _THEIRS = 'his';
    _THEM = 'him';
}

let storyText = `Once upon a time there was a ${_KID} named
    Jamie. Jamie liked going to the lake with ${_THEIR} family.
    One day, ${_THEY} decided to try to swim all the way across.`
```

Accessing child's languages

Here is a snippet showing how you might access information in the child's `languageList`:

```
// child.get('languageList') returns a string like 'en' or 'en my';
// transform to a list of two-letter codes like ['en'] or ['en', 'my']
let languageList = child.get('languageList').split(' ');
if (!languageList.length) {
    // Empty list of languages - no language data stored, possibly because
    // family registered before this was included in the child form.
    // Depending on study might include language survey in this case.
} else if (languageList.includes('es')) {
    // Child hears at least Spanish
} else {
    // Child has language data but is not exposed to Spanish
}
```

Accessing child's conditions

Here is a snippet showing how you might access information in the child's `conditionList`:

```javascript
let conditionList = child.get('conditionList').split(' ');
if (conditionList.includes('autism_spectrum_disorder')) {
    // child identified as having ASD
} else {
    // otherwise...
}
```

FEATURES COMMON TO ALL FRAMES

There are some features that are common to all frames, like being able to set a `generateProperties` function to adjust parameters on-the-fly based on what the child has done so far:

2.1 All frames support. . .

All Lookit frames share some common features. While frame-specific features are described on the pagse for those frames, like exp-lookit-video, you can also use any of the parameters listed here to customize any frame, and will receive the data and events described here.

2.1.1 Parameters

There are several parameters that ALL frames accept to allow you to customize the study "flow," which are:

selectNextFrame [String] Function to select which frame index to go to when using the 'next' action on this frame. Allows flexible looping / short-circuiting based on what has happened so far in the study (e.g., once the child answers N questions correctly, move on to next segment). Must be a valid Javascript function, returning a number from 0 through frames.length - 1, provided as a string.

Arguments that will be provided are: `frames`, `frameIndex`, `expData`, `sequence`, `child`, `pastSessions`

`frames` is an ordered list of frame configurations for this study; each element is an object corresponding directly to a frame you defined in the JSON document for this study (but with any randomizer frames resolved into the particular frames that will be used this time).

`frameIndex` is the index in `frames` of the current frame

`expData` is an object consisting of `frameId: frameData` pairs; the data associated with a particular frame depends on the frame kind.

`sequence` is an ordered list of frameIds, corresponding to the keys in `expData`.

`child` is an object that has the following properties - use child.get(propertyName) to access:

- `additionalInformation`: String; additional information field from child form

- `ageAtBirth`: String; child's gestational age at birth in weeks. Possible values are "24" through "39", "na" (not sure or prefer not to answer), "<24" (under 24 weeks), and "40>" (40 or more weeks).

- `birthday`: timestamp in format "Mon Apr 10 2017 20:00:00 GMT-0400 (Eastern Daylight Time)"

- `gender`: "f" (female), "m" (male), "o" (other), or "na" (prefer not to answer)

- `givenName`: String, child's given name/nickname

- `id`: String, child UUID

`pastSessions` is a list of previous response objects for this child and this study, ordered starting from most recent (at index 0 is this session!). Each has properties (access as pastSessions[i].get(propertyName)):

- `completed`: Boolean, whether they submitted an exit survey

- `completedConsentFrame`: Boolean, whether they got through at least a consent frame

- `conditions`: Object representing any conditions assigned by randomizer frames

- `createdOn`: timestamp in format "Thu Apr 18 2019 12:33:26 GMT-0400 (Eastern Daylight Time)"

- `expData`: Object consisting of frameId: frameData pairs

- `globalEventTimings`: list of any events stored outside of individual frames - currently just used for attempts to leave the study early

- `sequence`: ordered list of frameIds, corresponding to keys in expData

Example that just sends us to the last frame of the study no matter what: `"function(frames, frameIndex, frameData, expData, sequence, child, pastSessions) {return frames.length - 1;}"`

Example that just sends us to the next frame no matter what: `"function(frames, frameIndex, frameData, expData, sequence, child, pastSessions) {return frameIndex + 1;}"`

generateProperties [String] Function to generate additional properties for this frame (like {"kind": "exp-lookit-text"}) at the time the frame is initialized. Allows behavior of study to depend on what has happened so far (e.g., answers on a form or to previous test trials). Must be a valid Javascript function, returning an object, provided as a string.

Arguments that will be provided are: `expData`, `sequence`, `child`, `pastSessions`, `conditions`.

`expData`, `sequence`, and `conditions` are the same data as would be found in the session data shown on the Lookit experimenter interface under 'Individual Responses', except that they will only contain information up to this point in the study:

- `expData` is an object consisting of `frameId`: `frameData` pairs; the data associated with a particular frame depends on the frame kind.

- `sequence` is an ordered list of frameIds, corresponding to the keys in `expData`.

- `conditions` is an object representing the data stored by any randomizer frames; each key is a `frameId` for a randomizer frame and data stored depends on the randomizer used.

- `child` is an object that has the following properties - use `child.get(propertyName)` to access:

  - `additionalInformation`: String; additional information field from child form

  - `ageAtBirth`: String; child's gestational age at birth in weeks. Possible values are "24" through "39", "na" (not sure or prefer not to answer), "<24" (under 24 weeks), and "40>" (40 or more weeks).

  - `birthday`: Date object

  - `gender`: "f" (female), "m" (male), "o" (other), or "na" (prefer not to answer)

  - `givenName`: String, child's given name/nickname

  - `id`: String, child UUID

  - `languageList`: String, space-separated list of languages child is exposed to (2-letter codes)

  - `conditionList`: String, space-separated list of conditions/characteristics of child from registration form, as used in criteria expression, e.g. "autism_spectrum_disorder deaf multiple_birth"

- `pastSessions` is a list of previous response objects for this child and this study, ordered starting from most recent (at index 0 is this session!). Each has properties (access as pastSessions[i].get(propertyName)):

  - `completed`: Boolean, whether they submitted an exit survey

  - `completedConsentFrame`: Boolean, whether they got through at least a consent frame

  - `conditions`: Object representing any conditions assigned by randomizer frames

  - `createdOn`: Date object

  - `expData`: Object consisting of frameId: frameData pairs

  - `globalEventTimings`: list of any events stored outside of individual frames - currently just used for attempts to leave the study early

  - `sequence`: ordered list of frameIds, corresponding to keys in expData

  - `isPreview`: Boolean, whether this is from a preview session (possible in the event this is an experimenter's account)

Example:

```
function(expData, sequence, child, pastSessions, conditions) {
 return {
    'blocks':
        [
            {
                'text': 'Name: ' + child.get('givenName')
            },
            {
                'text': 'Frame number: ' + sequence.length
            },
            {
                'text': 'N past sessions: ' + pastSessions.length
            }
        ]
    };
}
```

Note: This example is split across lines for readability; when added to JSON it would need to be on one line.

parameters An object containing values for any parameters (variables) to use in this frame. Any property VALUES in this frame that match any of the property NAMES in *parameters* will be replaced by the corresponding parameter value. For details, see *Frame parameters*.

There are also some miscellaneous parameters you can set on any frame:

displayFullscreenOverride [Boolean | `false`] Set to *true* to display this frame in fullscreen mode, even if the frame type is not always displayed fullscreen. (For instance, you might use this to keep a survey between test trials in fullscreen mode.)

startSessionRecording [Boolean | `false`] Whether to start a session (multi-frame) recording as soon as possible upon loading this frame. It is recommended to use the dedicated frame exp-lookit-start-recording to start a session recording instead of adding this to an arbitrary frame.

Session recording allows you to to conduct video recording across multiple frames, simply specifying which frame to start and end on. Individual frames may also provide frame-specific recording capabilities; it is best NOT to conduct both a multiframe 'session' recording and frame-specific recording simultaneously as multiple video streams will eat up bandwidth. If you decide to use session recording, turn off recording for any frames that would otherwise record. There can be multiple session recordings in an experiment, e.g. from frames 1-3 and 5-10.

sessionMaxUploadSeconds: [Number | `10`] Maximum time allowed for whole-session video upload before proceeding, in seconds. Only used if `endSessionRecording` is true. Can be overridden by researcher, based on tradeoff between making families wait and losing data.

endSessionRecording [Boolean | `false`] Whether to end any session (multi-frame) recording at the end of this frame. It is recommended to use the dedicated frame exp-lookit-stop-recording to stop a session recording instead of adding this to an arbitrary frame.

2.1.2 Data collected

generatedProperties Any properties generated via a custom generateProperties function provided to this frame (e.g., a score you computed to decide on feedback). In general will be null.

frameDuration Duration between frame being inserted and call to `next`

frameType Type of frame: `EXIT` (exit survey), `CONSENT` (consent or assent frame), or `DEFAULT`

eventTimings Ordered list of events captured during this frame (oldest to newest). See "Events recorded" below as well as events specific to the particular frame type.

2.1.3 Events recorded

Events recorded by a frame will be available inside the `expData` for this session and frame. If the frame ID is `'0-video-config'`, then you could find a list of events in `expData['0-video-config']['eventTimings']`.

Each event is an object with at least the properties:

> **eventType** the name of the event - like `'nextFrame'` below
>
> **timestamp** the time when the event happened

Some events may have additional properties, which will be listed under the event description on the relevant frame.

The events recorded by the base frame are:

nextFrame When moving to next frame

previousFrame When moving to previous frame

2.2 Frame parameters

2.2.1 Overview

Frame parameters can be used to reuse or randomize values in any Lookit frame.

Rather than inserting actual values for frame properties such as stimulus image locations, you may want sometimes want to use a variable the way you would in a programming language - for instance, so that you can show the same cat picture throughout a group of frames, without having to replace it in ten separate places if you decide to use a different one. You can accomplish this (and more, including selecting randomly from or cycling through lists of values) by setting the `"parameters"` property on any frame (including frame groups and randomizers).

2.2.2 Syntax

You can pass frame parameters to any frame by including `"parameters": { ... }` in the frame definition, like this:

```
{
    'kind': 'FRAME_KIND',
    'parameters': {
        'FRAME_KIND': 'exp-lookit-text'
    }
}
```

Any property *values* in this frame that match any of the property *names* in `parameters` will be replaced by the corresponding parameter value. For example, the frame above will be resolved to have `'kind': 'exp-lookit-text'`.

Frame parameters are useful if you need to repeat values for different properties or across multiple frames, especially if your frame is actually a randomizer or group. You may use parameters nested within objects (at any depth) or within lists.

You can also use selectors to randomly sample from or permute a list defined in `parameters`. Suppose `STIMLIST` is defined in `parameters`, e.g. a list of potential stimuli. Rather than just using `STIMLIST` as a value in your frames, you can also:

- Select the Nth element (0-indexed) of the value of `STIMLIST`: (Will cause error if `N >= THELIST.length`)

  `'parameterName': 'STIMLIST#N'`

- Select (uniformly) a random element of the value of `STIMLIST`:

  `'parameterName': 'STIMLIST#RAND'`

- Set `parameterName` to a random permutation of the value of `STIMLIST`:

  `'parameterName': 'STIMLIST#PERM'`

- Select the next element in a random permutation of the value of `STIMLIST`, which is used across all substitutions in this randomizer. This allows you, for instance, to provide a list of possible images in your `parameterSet`, and use a different one each frame with the subset/order randomized per participant. If more `STIMLIST#UNIQ` parameters than elements of `STIMLIST` are used, we loop back around to the start of the permutation generated for this randomizer.

  `'parameterName': 'STIMLIST#UNIQ'`

2.2.3 Case study: randomizing the order of options in a survey

Suppose you're including a survey where you ask participants to record whether their child performed a certain task, and you want to present the options in a random order to avoid systematically biasing the results towards either option. You start with a survey frame like this (see the frame docs for more information about this frame):

```
"example-survey": {
    "kind": "exp-lookit-survey",
    "formSchema": {
        "schema": {
            "type": "object",
            "title": "And now, a thrilling survey!",
            "properties": {
                "didit": {
                    "enum": ["yes", "no"],
```

(continues on next page)

```
                "type": "string",
                "title": "Did your child do the thing?",
                "default": ""
            }
        }
    },
    "options": {
        "fields": {
            "didit": {
                "type": "radio",
                "validator": "required-field"
            }
        }
    }
}
},
```

To randomize the options, we'll need to make a few small changes. First, add `"sort":   false` to the options for your `didit` field, so that AlpacaJS doesn't automatically sort the options alphabetically.

Next, you want the `enum` list for `didit` to actually be in random order. To achieve that, you can add a property like `DIDIT_OPTIONS` as a frame property, and then specify that the value of `enum` should be a random permutation of that list, like this:

```
"example-survey": {
    "kind": "exp-lookit-survey",
    "formSchema": {
        "schema": {
            "type": "object",
            "title": "And now, a thrilling survey!",
            "properties": {
                "didit": {
                    "enum": "DIDIT_OPTIONS#PERM",
                    "type": "string",
                    "title": "Did your child do the thing?",
                    "default": ""
                }
            }
        },
        "options": {
            "fields": {
                "didit": {
                    "sort": false,
                    "type": "radio",
                    "validator": "required-field"
                }
            }
        }
    },
    "parameters": {
        "DIDIT_OPTIONS": ["yes", "no"]
    }
},
```

2.3 Frame groups

2.3.1 Overview

Sometimes it may be convenient to group several frames together. For instance, you might want to randomize the order of several blocks of trials, but keep the blocks together.

To create a frame group, set the frame `"kind"` to `"group"`.

You will also need to provide a `"frameList"` which is a list of frames that go in this group.

You can optionally provide a `"commonFrameProperties"` object which provides default parameter-value pairs to add to each frame in the list (any parameters additionally defined in the `frameList` will take precedence).

As with other frames, `"parameters"` can be defined on the frame group to allow substitution of values. This can be a convenient way to substitute in common values across several frames, or even across your entire study - for instance, if you want to use the same `baseDir` in every frame that supports it.

2.3.2 Example

Here is an example of a frame group that just contains two text frames:

```
"testFrameGroup": {
    "kind": "group",
    "frameList": [
        {
            "id": "first-test-trial",
            "blocks": [
                {
                    text: "Hello and welcome to the study"
                }
            ]
        },
        {
            "id": "second-test-trial",
            "blocks": [
                {
                    text: "Some more info"
                }
            ]
        }
    ],
    "commonFrameProperties": {
        "kind":  "exp-lookit-text"
    }
}
```

2.3.3 Parameters

frameList [Array] List of frames to be included in this group. All frames will be displayed in order. Each frame is an object with any necessary frame-specific properties specified. The 'kind' of frame can be specified either here (per frame) or in commonFrameProperties. If a property is defined for a given frame both in this frame list and in commonFrameProperties, the value in the frame list will take precedence.

(E.g., you could include `'kind':` `'normal-frame'` in commmonFrameProperties, but for a single frame in frameOptions, include `'kind':` `'special-frame'`.)

commonFrameProperties [Object] Object describing common parameters to use as defaults for every frame created by this randomizer. Parameter names and values are as described in the documentation for the frameType used.

2.4 Conditional logic

In some cases, what happens next in your study will need to depend on what has happened so far, what happened during previous sessions of the study, and/or information about the participant. For instance, perhaps you want to move on from a training segment after the participant answers three questions in a row correctly, or you want to start with an eligibility survey and only route people to the rest of the study if they meet detailed criteria. Or maybe you just want to personalize instructions or stimuli with the child's name and gender! All Lookit frames allow you to provide either or both of the following properties to flexibly specify conditional behavior:

1. *generateProperties*: A function that takes `expData`, `sequence`, `child`, `pastSessions`, and `conditions` objects, and returns an object representing any additional properties that should be used by this frame - e.g., the frame type, text blocks, whether to do recording, etc. This is called when the frame is initialized.

2. selectNextFrame: A function that takes `frames`, `frameIndex`, `expData`, `sequence`, `child`, and `pastSessions` and returns that frame index to go to when using the 'next' action on this frame. For instance, this allows you to skip to the end of the study (or a frame of a particular type) if the child has gotten several questions correct. This function is called upon proceeding to the next frame, so it has access to data collected on this frame.

Each of these properties is specified as a string, which must define a Javascript function. Formal documentation for these properties is linked above.

2.4.1 Writing your functions

In practice, if you want to add some conditional behavior and are wondering e.g. how to get the child's first name or birthday, or how to determine what condition the child is in, it may be easiest to get started by adding a dummy function like the following to the frame in question:

```
"generateProperties": "function(expData, sequence, child, pastSessions, conditions)
→{console.log(expData); console.log(sequence); console.log(child); console.
→log(pastSessions); console.log(conditions); return {};}"

"selectNextFrame": "function(frames, frameIndex, frameData, expData, sequence, child,
→pastSessions) {console.log(frames); console.log(frameIndex); console.log(frameData);
→ console.log(expData); console.log(sequence); console.log(child); console.
→log(pastSessions); return (frameIndex + 1);}"
```

These functions just log each of the arguments they're given the Javascript console; there you can take a look and play around with how you'd access and manipulate the properties you need. The `generateProperties` function above just return an empty object, not assigning any properties. The `selectNextFrame` function just returns

`frameIndex + 1`, i.e. says the next frame should be the one after this one, which doesn't change the frame's regular behavior.

2.4.2 Adding and removing line breaks as you write

Although you'll need to enter these properties as single-line strings in the Lookit study editor, they are obviously not very readable that way! You can go from a single-line string back to something readable using a Javascript 'beautifier' like this - you may want to do that to better understand the examples below. When you are writing your own functions, you can write them on multiple lines in your text editor and then either strip out the line breaks using your text editor or one of many online tools like this.

2.4.3 Example: eligibility survey

Here is an example of a situation where you might want to determine the sequence of frames in a study and/or behavior of those frames based on data collected earlier in the study. Suppose you want to start off with a survey to determine eligibility, using criteria that go beyond what is available in Lookit child/demographic surveys and usable for automatic eligibility detection. (Perhaps your study is very involved or won't make sense to people who don't meet criteria, so you don't want to just have everyone participate and filter the data afterwards.)

A similar approach would be appropriate if you wanted to customize the behavior of the study based on user input - e.g., using the child's favorite color for stimuli, let the family choose which game they want to play this time, or let the family choose whether to 'actually' participate (and have video recorded) or just see a demo.

This example has three top-level frames: an eligibility survey, a study procedure (which depends on eligibility as determined from the survey), and an exit survey (with debriefing text that depends on eligibility too).

```
{
    "frames": {
        "exit-survey": {
            "kind": "exp-lookit-exit-survey",
            "generateProperties": "function(expData, sequence, child, pastSessions)
↪{var eligible = expData['1-study-procedure']['generatedProperties']['ELIGIBLE']; if␣
↪(eligible) { return { 'debriefing': {                    'text': 'In this study, we␣
↪were looking at why babies love cats. Your child actually participated. A real␣
↪debriefing would be more detailed.', 'title': 'Thank you!' } }; } else { return {
↪'debriefing': {                    'text': 'In this study, we would have looked at why␣
↪your child loved cats. Your child did not actually participate though. A real␣
↪debriefing would make more sense.', 'title': 'Thank you!' } }; }}"
        },
        "eligibility-survey": {
            "kind": "exp-lookit-survey",
            "formSchema": {
                "schema": {
                    "type": "object",
                    "title": "Eligibility survey",
                    "properties": {
                        "nCats": {
                            "type": "integer",
                            "title": "How many cats do you have?",
                            "maximum": 200,
                            "minimum": 0,
                            "required": true
                        },
                        "loveCats": {
                            "enum": [
```

(continues on next page)

```
                        "yes",
                        "no"
                    ],
                    "type": "string",
                    "title": "Does your baby love cats?",
                    "required": true
                }
            }
        },
        "options": {
            "fields": {
                "nCats": {
                    "numericEntry": true
                },
                "loveCats": {
                    "type": "radio",
                    "message": "Please answer this question.",
                    "validator": "required-field"
                }
            }
        }
    },
    "nextButtonText": "Continue"
},
"study-procedure": {
    "kind": "exp-frame-select",
    "frameOptions": [
        {
            "kind": "exp-frame-select",
            "frameOptions": [
                {
                    "kind": "exp-lookit-text",
                    "blocks": [
                        {
                            "emph": true,
                            "text": "Let's start the study!"
                        },
                        {
                            "text": "Some info about cats..."
                        }
                    ]
                },
                {
                    "kind": "exp-lookit-text",
                    "blocks": [
                        {
                            "emph": true,
                            "text": "Cats are great"
                        },
                        {
                            "text": "We are measuring how much your child␣
→loves cats now. Beep boop!"
                        }
                    ]
                }
            ]
        },
```

```
                    {
                        "kind": "exp-lookit-text",
                        "blocks": [{
                                "emph": true,
                                "text": "Your child is not eligible for this study"
                        },
                        {
                                "text": "Either you do not have any cats or your child
→does not love cats."
                        }
                        ]
                    }
                ],
                "generateProperties": "function(expData, sequence, child, pastSessions)
→{var formData = expData['0-eligibility-survey'].formData; if (formData.nCats >= 1 &&
→ formData.loveCats == 'yes') { console.log('eligible'); return { 'whichFrames': 0,
→'ELIGIBLE': true } } else { console.log('ineligible'); return { 'whichFrames': 1,
→'ELIGIBLE': false } } }"
        }
    },
    "sequence": [
        "eligibility-survey",
        "study-procedure",
        "exit-survey"
    ]
}
```

Here's how it works:

1. The study procedure is set up as *exp-frame-select* frame, and we decide on-the-spot which of the two `frameOptions` to use based on the data in the survey by providing a `generateProperties` function that returns a value for `whichFrames`. The function `generateProperties` is called when we get to the `study-procedure` frame, and the key-value pairs it returns get added to the other parameters for this frame (like `kind` and `frameOptions`). In this case, it checks to see whether the survey says the family has at least one cat *and* the child loves cats; in that case, the child is eligible to participate.

   Additionally, the object `generateProperties` returns is stored under the key `generatedProperties` in expData for this frame, so that we can use the output later. That's why we also include either `'ELIGIBLE': true` or `'ELIGIBLE': false` - that way we can reuse this determination later on in another `generateProperties` function.

2. If the child isn't eligible, the `study-procedure` frame just resolves to a single `exp-lookit-text` frame, at index 1 of `frameOptions`. If the child is eligible, the `study-procedure` frame resolves to a second `exp-frame-select` frame, which just serves to bundle up a few text frames. We don't provide `whichFrames`, so all of the `frameOptions` listed will be shown in order. (We could also have set this up without a nested `exp-frame-select` frame, e.g. by putting all three `exp-lookit-text` frames in the outer `frameOptions` and saying that if the child is eligible, use `whichFrames = [0, 1]`, and if not, `whichFrames = 2`.)

3. After the study procedure is done, everyone goes to an exit survey. The `generateProperties` function of the exit survey returns different debriefing text based on the stored `ELIGIBLE` value we defined earlier.

Note that the data stored in `expData`` will include frame data for the `exp-frame-select` frames, even though these are not actually displayed as frames separate from the contents they resolve to. For a child who is eligible, the keys in `expData` will be:

   • `0-eligibility-survey`

- `1-study-procedure` (the outer `exp-frame-select` frame)

- `1-study-procedure-0` (the inner `exp-frame-select` frame)

- `1-study-procedure-0-0` (the first `exp-lookit-text` frame)

- `1-study-procedure-0-1` (the second `exp-lookit-text` frame)

2.4.4 Example: skipping a survey if it was completed previously

Suppose your list of frames includes `instructions`, `eligibility-survey`, `detailed-survey`, and `test-trial`, in that order. You want to show all of these frames in order in general (although you'll skip straight from eligibility-survey to test-trial if the person completing the study is not eligible to complete the detailed-survey). But if someone has already completed the detailed-survey, you want to skip straight from instructions to test-trial. You can do that by adding the following to the JSON specification for the instructions frame:

```
"selectNextFrame": "function(frames, frameIndex, frameData, expData, sequence, child,␣
→pastSessions) {if (pastSessions.some(sess => Object.keys(sess.get('expData', {})).
→some(frId => frId.endsWith('-detailed-survey')))) {return frameIndex + 3;} else
→{return frameIndex + 1;}}"
```

What this does is check to see if the `pastSessions` data contains any session with expData for a `*-detailed-survey` frame. If so, it sets the "next" frame to this frame + 3 - i.e., instead of incrementing by 1, it increments by 3, so it skips the two survey frames.

2.4.5 Example: waiting for successful training

Sometimes, you might want to skip ahead to the next section of an experiment once certain criteria are met. For instance:

- you might have a study where questions get harder and harder over time, and you just want to keep asking until the child gets N wrong in a row

- you might want to have a "training" section that allows the family to practice until they're ready

- you might want to make one section of a study optional, and skip over it if the parent opts to (or if it's not applicable to them)

Here's an example study where we wait for the child to get two "training" questions right, then proceed to a "test" question:

```
{
    "frames": {
        "exit-survey": {
            "kind": "exp-lookit-exit-survey",
            "debriefing": {
                "title": "Thank you!",
                "text": "Thank you for participating in this study"
            }
        },
        "training-question-block": {
            "kind": "exp-frame-select",
            "frameOptions": [
                {}, {}, {}, {}, {}, {}, {}, {}, {}, {}
            ],
            "commonFrameProperties": {
                "kind": "exp-lookit-survey",
```

(continues on next page)

```
                "generateProperties": "  function(expData, sequence, child,␣
→pastSessions) {                  var n = Math.floor(Math.random() * Math.floor(20)); ␣
→            var m = Math.floor(Math.random() * Math.floor(20));                 ␣
→return {                         'formSchema': {                      'schema': {     ␣
→                'type': 'object',                                  'title': 'Math␣
→practice question',                         'properties': {                          ␣
→        'add': {                                       'enum': [                       ␣
→                'low',                                      'correct',              ␣
→                          'high'                             ],      'title
→': 'What is ' + n + ' plus ' + m + '?',                              'required
→': true                     }                             }      ␣
→         },                              'options': {                        'fields
→': {                          'add': {
→'type': 'radio',    'optionLabels': [n + m - 1, n + m, n + m + 1],              ␣
→            'message': 'Please answer this question.',                      ␣
→         'validator': 'required-field'}}}}}}",
                "selectNextFrame": "function(frames, frameIndex, frameData, expData,␣
→sequence, child, pastSessions) {    var testFrame = 0; for (var iFrame = 0; iFrame
→< frames.length; iFrame++) {if (frames[iFrame]['id'].indexOf('test-question') != -
→1) {testFrame = iFrame; break;}} if ((sequence.length >= 3) &&␣
→(expData[sequence[sequence.length - 2]]['formData']['add'] == 'correct' ) &&␣
→(expData[sequence[sequence.length - 1]]['formData']['add'] == 'correct')){    ␣
→return testFrame;    }    else {        return frameIndex + 1;    }}"
            }
        },
        "test-question": {
            "kind": "exp-lookit-survey",
            "generateProperties": "  function(expData, sequence, child, pastSessions)
→{             var n = Math.floor(Math.random() * Math.floor(20));                 ␣
→var m = Math.floor(Math.random() * Math.floor(20));                 return {       ␣
→        'formSchema': {                      'schema': {                      ␣
→    'type': 'object',                                  'title': 'Math test question',   ␣
→                'properties': {                                  'subtract':
→{                                  'enum': [                                   ␣
→  'low',                                      'correct',                        ␣
→          'high'                             ],    'title': 'What is ' +␣
→n + ' minus ' + m + '?',                                'required': true       ␣
→                }                             }                      },  ␣
→                'options': {                                  'fields': {       ␣
→                'subtract': {                                  'type': 'radio',
→   'optionLabels': [n - m - 1, n - m, n - m + 1],              ␣
→'message': 'Please answer this question.',
→'validator': 'required-field'}}}}}}"
        }
    },
    "sequence": [
        "training-question-block",
        "test-question",
        "exit-survey"
    ]
}
```

There are three sections in the study: a block of up to 10 training questions, a single test question, and an exit
survey. We use an `exp-frame-select` frame to quickly create ten identical training question frames, by putting
all of the frame properties into `commonFrameProperties`. We use `generateProperties` not to do anything
contingent on the child or study data, but just to programmatically generate the questions - this way we can choose
random numbers for each question. Finally, we add a `selectNextFrame` function to the training questions. Let's

take a closer look at that function:

```javascript
function(frames, frameIndex, frameData, expData, sequence, child, pastSessions) {
    // First, find the index of the test frame in case we need to go there
    var testFrame = 0;
    for (var iFrame = 0; iFrame < frames.length; iFrame++) {
        if (frames[iFrame]['id'].indexOf('test-question') != -1) {
            testFrame = iFrame;
            break;
        }
    }
    // If the last two questions were answered correctly, go to test
    if ((sequence.length >= 3) && (expData[sequence[sequence.length - 2]]['formData'][
→'add'] == 'correct') && (expData[sequence[sequence.length - 1]]['formData']['add']␣
→== 'correct')) {
        return testFrame;
    } else {
    // Otherwise, just go to the next frame
        return frameIndex + 1;
    }
}
```

We first use the list of `frames` to identify the index of the test question. (In this case we could safely assume it's the second-to-last frame, too. But in a more complex experiment, we might want to find it like this.)

Then we check whether (a) there are already at least 3 frames including this one in the `sequence` (two practice questions plus the initial `exp-frame-select` frame) and (b) the last two questions including this one were answered correctly. If so, we skip right to the test question!

2.4.6 Example: personalized story

One of the objects you have access to in your `generateProperties` function is the `child`. This allows you to use child data in selecting stimuli, instructions, or procedures. A simple use case would be personalizing a story (or instructions) using the child's name and gender. Here's an example:

```json
{
    "frames": {
        "personalized-story": {
            "kind": "exp-lookit-text",
            "generateProperties": "function(expData, sequence, child, pastSessions,␣
→conditions) {var childName = child.get('givenName'); var genderedChild; if (child.
→get('gender') == 'f') {    genderedChild = 'girl';} else if (child.get('gender') ==
→ 'm') {    genderedChild = 'boy';} else {genderedChild = 'kiddo';} var line1 = 'Once␣
→upon a time, there was a little ' + genderedChild + ' named ' + childName + '.';␣
→var line2 = childName + ' loved to draw.'; return {'blocks': [{'text': line1}, {
→'text': line2}]};}"
        }
    },
    "sequence": [
        "personalized-story"
    ]
}
```

2.4.7 Example: debriefing text that depends on experimental condition

One fairly common and straightforward use case for customizing frames based on data from the experiment is that you might like to debrief parents at the end of the study based on the experimental condition their child was in, just like you would in the lab.

Here's an example where we have an experimental "procedure" that depends on condition assignment in a `random-parameter-set` frame, and mention the condition in the debriefing text:

```
{
    "frames": {
        "exit-survey": {
            "kind": "exp-lookit-exit-survey",
            "debriefing": {
                "title": "Thank you!",
                "text": "Thank you for participating in this study. Your child was in␣
↪the "
            },
            "generateProperties": "function(expData, sequence, child, pastSessions,␣
↪conditions) {if (conditions['1-study-procedure']['conditionNum'] == 0) {return {
↪'debriefing': {'title': 'Thank you!', 'text': 'Your child was in the cats condition.
↪'}};} else  {return {'debriefing': {'title': 'Thank you!', 'text': 'Your child was␣
↪in the dogs condition.'}};} }"
        },
        "study-procedure": {
            "sampler": "random-parameter-set",
            "kind": "choice",
            "frameList": [
                {
                    "kind": "exp-lookit-text",
                    "blocks": [
                        {
                            "text": "PROCEDURE_TEXT",
                            "title": "PROCEDURE_TITLE"
                        }
                    ]
                }
            ],
            "parameterSets": [
                {
                    "PROCEDURE_TEXT": "All about cats",
                    "PROCEDURE_TITLE": "Cats say meow!"
                },
                {
                    "PROCEDURE_TEXT": "All about dogs",
                    "PROCEDURE_TITLE": "Dogs say woof!"
                }
            ]
        }
    },
    "sequence": [
        "study-procedure",
        "exit-survey"
    ]
}
```

Your debriefing information could also take into account other factors - for instance, if you were conducting a give-N task, you could actually give an automatic estimate of the child's knower-level or show a chart of their responses! As

an exercise, try personalizing the debriefing text to use the child's name.

SPECIFIC FRAMES

These are you can use in your Lookit study, like instructions pages or looking-time trials. The documentation will tell you how each frame works, what data it collects, and what parameters you need to give it.

3.1 exp-frame-select

3.1.1 Overview

Frame that allows you to specify a list of possible frames to show, plus an index or list of indices of which ones to actually show.

The frame(s) will be inserted into the sequence of frames for this study on the fly, so that you can use a custom *generateProperties* function to select which frame(s) to show. (For more information on making study behavior conditional on data collected, see conditional_logic.)

This frame serves as a wrapper for the randomizer *select*, which is evaluated during experiment parsing and cannot be modified on the fly.

Warning: no `selectNextFrame` available

To avoid unpredictable behavior, this frame does not itself use any selectNextFrame passed to it. (Frames *within* the `frameOptions` list are welcome to make use of `selectNextFrame`, though!)

Finding data from frames created by exp-lookit-select

Data will be stored for this frame so that any `generatedProperties` are available for future use; however, it will proceed immediately upon loading to the first frame that is specified (or the next frame in the original sequence, if it turns out that *whichFrames* is an empty list).

In `expData`, the frame keys for all frames generated by this frame will be prefixed by this frame's ID, with an index within `whichFrames` appended to the end of the ID. For instance, if this frame's ID is `1-study-procedure`, and it generates three frames, we would have keys `1-study-procedure`, `1-study-procedure-0`, `1-study-procedure-1`, and `1-study-procedure-2`.

More general functionality

Below is information specific to this particular frame. There may also be available parameters, events recorded, and data collected that come from the following more general sources:

- the *base frame* (things all frames do)

3.1.2 Examples

This frame would show one frame if the child is eligible to participate based on a previous survey, and another if not. (Note: this is just an example to show how you could use this frame. It will not work if you copy and paste it without the eligibility survey the "generateProperties" function is referencing!)

```
"study-procedure": {
    "kind": "exp-frame-select",
    "frameOptions": [
        {
            "kind": "exp-lookit-text",
            "blocks": [
                {
                    "emph": true,
                    "text": "Cats are great"
                },
                {
                    "text": "We are measuring how much your child loves cats now.
→Beep boop!"
                }
            ]
        },
        {
            "kind": "exp-lookit-text",
            "blocks": [{
                    "emph": true,
                    "text": "Your child is not eligible for this study"
                },
                {
                    "text": "Either you do not have any cats or your child does not
→love cats."
                }
            ]
        }
    ],
    "generateProperties": "function(expData, sequence, child, pastSessions) {var
→formData = expData['0-eligibility-survey'].formData; if (formData.nCats >= 1 &&
→formData.loveCats == 'yes') { console.log('eligible'); return { 'whichFrames': 0,
→'ELIGIBLE': true } } else { console.log('ineligible'); return { 'whichFrames': 1,
→'ELIGIBLE': false } } }"
}
```

3.1.3 Parameters

frameOptions [Array | [ ]] List of frames that can be created by this randomizer. Each frame is an object with any necessary frame-specific properties specified. The 'kind' of frame can be specified either here (per frame) or in commonFrameProperties. If a property is defined for a given frame both in this frame list and in common-FrameProperties, the value in the frame list will take precedence.

(E.g., you could include 'kind': 'normal-frame' in commmonFrameProperties, but for a single frame in frameOptions, include 'kind': 'special-frame'.)

commonFrameProperties [Object | { }] Object describing common parameters to use in EVERY frame created by this randomizer. Parameter names and values are as described in the documentation for the frameType used.

whichFrames [Array or Number | −1] Index or indices (0-indexed) within frameOptions to actually use. This can be either a number (e.g., 0 or 1 to use the first or second option respectively) or an array providing an ordered list of indices to use (e.g., [0, 1] or [1, 0] to use the first then second or second then first options, respectively). All indices must be integers >= 0 and < frameOptions.length.

If not provided or -1, the entire frameOptions list is used in order. (If empty list is provided, however, that is respected and no frames are inserted by this randomizer.)

3.1.4 Data collected

No data is stored specifically by this frame.

3.1.5 Events recorded

No events are recorded specifically by this frame.

3.2 exp-lookit-calibration

3.2.1 Overview

Frame to do calibration for looking direction. Shows a small video/image in a sequence of locations so you'll have video of the child looking to those locations at known times.

The attention-grabber can be either a small video or an image (see examples below of each). Images can be animated (spinning or bouncing).

The image or video can be displayed at center, left, or right of the screen. You can specify the sequence of positions or use the default ['center', 'left', 'right', 'center']. Each time it moves, the video (if any) and audio restart, and an event is recorded with the location and time (and time relative to any video recording) of the segment start.

What it looks like
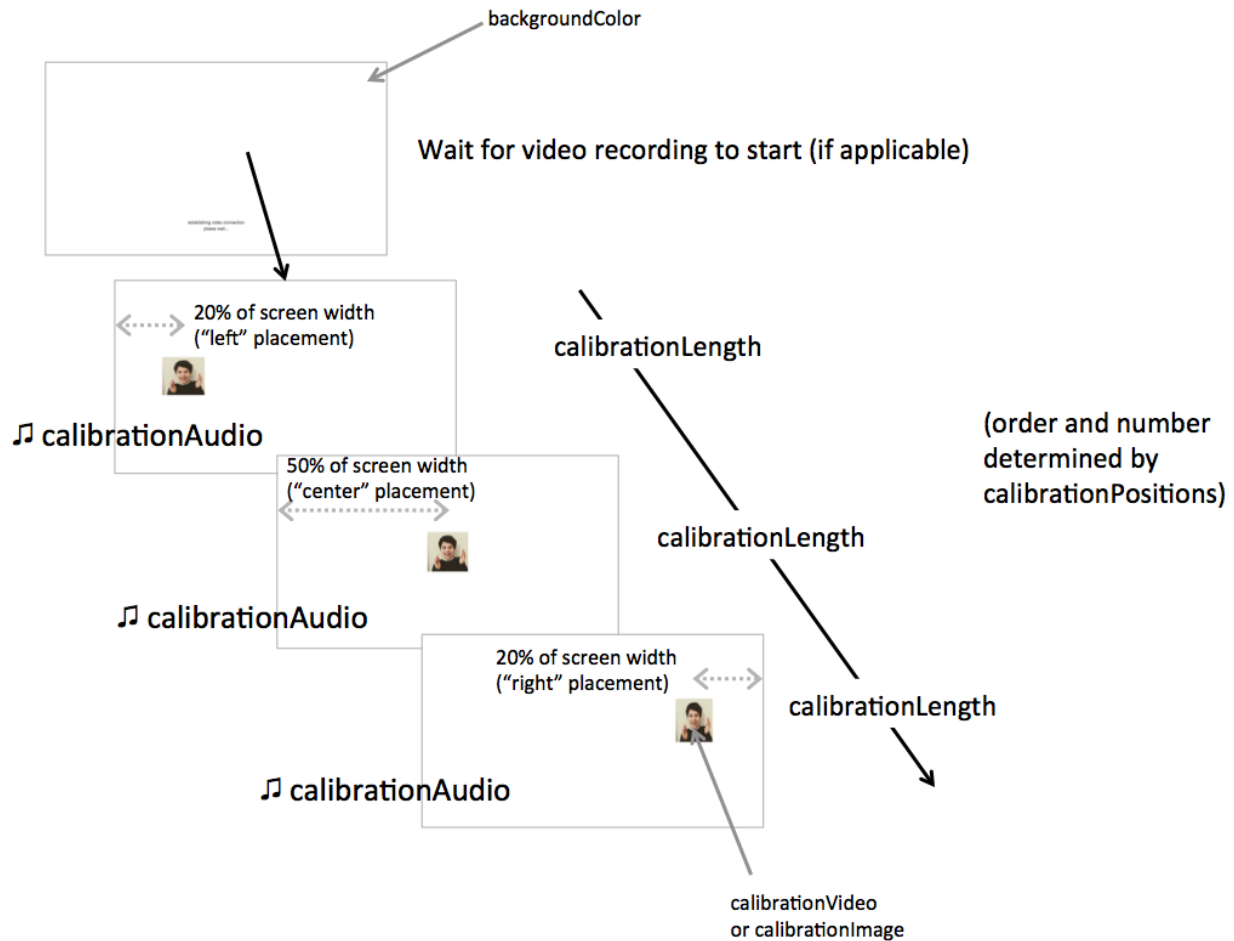
Recording

Generally you will want to have webcam video of this frame. You can set doRecording to true to make a video clip just for this frame. Recording will begin at the same time the first calibration stimulus is shown. Alternately, you can use session-level recording (set startSessionRecording to true on this or a previous frame). If either type of recording is starting on this frame, it waits until recording starts to display the first calibration segment.

Fullscreen display

This frame is displayed fullscreen, to match the frames you will likely want to compare looking behavior on. If the participant leaves fullscreen, that will be recorded as an event, and a large "return to fullscreen" button will be displayed. Don't use video coding from any intervals where the participant isn't in fullscreen mode - the position of the attention-grabbers won't be as expected.

If the frame before this is not fullscreen, that frame needs to include a manual "next" button so that there's a user interaction event to trigger fullscreen mode. (Browsers don't allow us to switch to FS without a user event.)

Specifying where files are

Several of the parameters for this frame can be specified either by providing a list of full URLs and file types, or by providing just a filename that will be interpreted relative to the `baseDir`. See the *expand-assets mixin* mixin that this frame uses.

More general functionality

Below is information specific to this particular frame. There may also be available parameters, events recorded, and data collected that come from the following more general sources:

- the *base frame* (things all frames do)

- *video-record mixin*

- *expand-assets mixin*

3.2.2 Examples

This frame will show an image at center, left, and right, along with chimes each time.

```
"calibration-with-image": {
    "kind": "exp-lookit-calibration",
    "baseDir": "https://www.mit.edu/~kimscott/placeholderstimuli/",
    "audioTypes": [
        "ogg",
        "mp3"
    ],
    "videoTypes": [
        "webm",
        "mp4"
    ],
    "calibrationImage": "peekaboo_remy.jpg",
    "calibrationLength": 3000,
    "calibrationPositions": [
        "center",
        "left",
        "right"
    ],
    "calibrationAudio": "chimes",
    "calibrationImageAnimation": "spin"
}
```

This frame will show a small video at center, left, and right, along with chimes each time.

```
"calibration-with-video": {
    "kind": "exp-lookit-calibration",
    "baseDir": "https://www.mit.edu/~kimscott/placeholderstimuli/",
    "audioTypes": [
        "ogg",
        "mp3"
    ],
    "videoTypes": [
        "webm",
        "mp4"
    ],
```

```
    "calibrationLength": 3000,
    "calibrationPositions": [
        "center",
        "left",
        "right"
    ],
    "calibrationAudio": "chimes",
    "calibrationVideo": "attentiongrabber"
}
```

3.2.3 Parameters

doRecording [Boolean | `true`] Whether to do any video recording during this frame. Default true. Set to false for e.g. last frame where just doing an announcement.

backgroundColor [String | `white`] Color of background. See CSS specs for acceptable syntax: can use color names ('blue', 'red', 'green', etc.), or rgb hex values (e.g. '#800080' - include the '#')

calibrationLength [Number | `3000`] Length of each calibration segment in ms

calibrationPositions [Array | `['center', 'left', 'right', 'center']`] Ordered list of positions to show calibration segment in. Options are "center", "left", "right". Ignored if calibrationLength is 0.

calibrationAudio [String or Array | `[]`] Audio to play when the attention-grabber is placed at each location (will be played once from the start, but cut off if it's longer than calibrationLength).

This can either be an array of *{src: 'url', type: 'MIMEtype'}* objects for calibration audio, or just a string to use the full URLs based on *baseDir*.

calibrationVideo [String or Array | `[]`] Calibration video (played from start at each calibration position). Supply either a calibration video or calibration image, not both.

This can be either an array of {src: 'url', type: 'MIMEtype'} objects or just a string like *attentiongrabber* to rely on the *baseDir* and *videoTypes* to generate full paths.

calibrationImage [String | `''`] Image to use for calibration - will be placed at each location. Supply either a calibration video or calibration image, not both.

This can be either a full URL or just the filename (e.g. "star.png") to use the full path based on *baseDir* (e.g. *baseDir/img/star.png*).

calibrationImageAnimation [String | `'spin'`] Which animation to use for the calibration image. Options are 'bounce', 'spin', or '' (empty to not animate).

3.2.4 Data collected

No data is recorded specifically by this frame type.

3.2.5 Events recorded

The events recorded specifically by this frame are:

startCalibration Beginning of each calibration segment

   location [String] The location of calibration image/video, relative to child: 'left', 'right', or 'center'

3.3 exp-lookit-change-detection

3.3.1 Overview

Frame for a preferential looking "alternation" or "change detection" paradigm trial, in which separate streams of images are displayed on the left and right of the screen.

Typically, on one side images would be alternating between two categories - e.g., images of 8 vs. 16 dots, images of cats vs. dogs - and on the other side the images would all be in the same category.

The frame starts with an optional brief "announcement" segment, where an attention-getter video is displayed and audio is played. During this segment, the trial can be paused and restarted.

You can customize the appearance of the frame: background color overall, color of the two rectangles that contain the image streams, and border of those rectangles. You can also specify how long to present the images for, how long to clear the screen in between image pairs, and how long the test trial should be altogether.

You provide four lists of images to use in this frame: *leftImagesA*, *leftImagesB*, *rightImagesA*, and *rightImagesB*. The left stream will alternate between images in *leftImagesA* and *leftImagesB*. The right stream will alternate between images in *rightImagesA* and *rightImagesB*. They are either presented in random order (default) within those lists, or can be presented in the exact order listed by setting *randomizeImageOrder* to false.

The timing of all image presentations and the specific images presented is recorded in the event data.

What it looks like
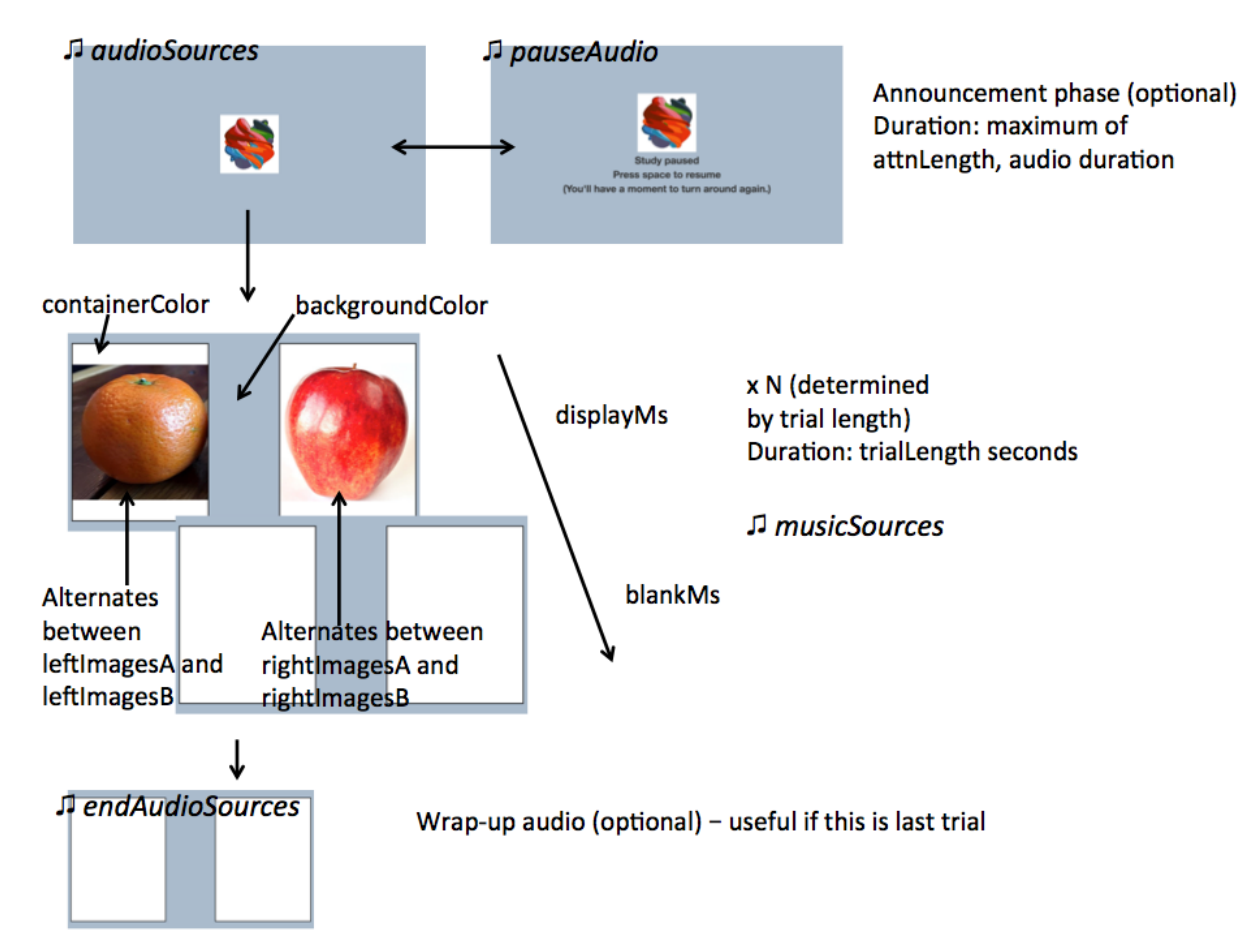
Recording

If `doRecording` is true (default), then we wait for recording to begin before the actual test trial can begin. We also always wait for all images to pre-load, so that there are no delays in loading images that affect the timing of presentation.

Display

This frame is displayed fullscreen; if the frame before it is not, that frame needs to include a manual "next" button so that there's a user interaction event to trigger fullscreen mode. (Browsers don't allow us to switch to FS without a user event.)

Specifying where files are

Several of the parameters for this frame can be specified either by providing a list of full URLs and file types, or by providing just a filename that will be interpreted relative to the `baseDir`. See the *expand-assets mixin* tool that this frame uses.

More general functionality

Below is information specific to this particular frame. There may also be available parameters, events recorded, and data collected that come from the following more general sources:

- the *base frame* (things all frames do)
- *video-record mixin*
- *expand-assets mixin*

3.3.2 Example

This frame will alternate between fruit and shapes on the left, and just fruit on the right.

```
"alt-trial": {
    "kind": "exp-lookit-change-detection",
    "baseDir": "https://www.mit.edu/~kimscott/placeholderstimuli/",
    "videoTypes": ["mp4", "webm"],
    "audioTypes": ["mp3", "ogg"],
    "trialLength": 15,
    "attnLength": 2,
    "fsAudio": "sample_1",
    "unpauseAudio": "return_after_pause",
    "pauseAudio": "pause",
    "videoSources": "attentiongrabber",
    "musicSources": "music_01",
    "audioSources": "video_01",
    "endAudioSources": "all_done",
    "border": "thick solid black",
    "leftImagesA": ["apple.jpg", "orange.jpg"],
    "rightImagesA": ["square.png", "tall.png", "wide.png"],
    "leftImagesB": ["apple.jpg", "orange.jpg"],
    "rightImagesB": ["apple.jpg", "orange.jpg"],
    "startWithA": true,
    "randomizeImageOrder": true,
    "displayMs": 500,
    "blankMs": 250,
    "containerColor": "white",
    "backgroundColor": "#abc"
}
```

3.3.3 Parameters

doRecording [Boolean | `true`] Whether to do webcam recording on this frame

attnLength [Number | `0`] minimum amount of time to show attention-getter in seconds. If 0, attention-getter segment is skipped.

trialLength [Number | `60`] length of alternation trial in seconds. This refers only to the section of the trial where the alternating image streams are presented - it does not count any announcement phase.

audioSources [String or Array | `[]`] Array of {src: 'url', type: 'MIMEtype'} objects for instructions during attention-getter video, OR string relative to `baseDir`. The entire audio file will play before moving on, even if it's longer than `attnLength`.

musicSources [String or Array | `[]`] Array of {src: 'url', type: 'MIMEtype'} objects, OR string relative to `baseDir`, for music during trial. This will loop for the duration of the trial.

endAudioSources [String or Array | `[]`] Array of {src: 'url', type: 'MIMEtype'} objects for audio, OR string relative to `baseDir`, to play after completion of trial (optional; used for last trial "okay to open your eyes now" announcement)

videoSources [String or Array | `[]`] Array of {src: 'url', type: 'MIMEtype'} objects for attention-getter video, OR string relative to `baseDir`. Will play in a loop for announcement phase.

pauseAudio [String or Array | `[]`] Array of {src: 'url', type: 'MIMEtype'} objects for audio to play upon pausing study, OR string relative to `baseDir`.

unpauseAudio [String or Array | `[]`] Array of {src: 'url', type: 'MIMEtype'} objects for audio to play upon un-pausing study, OR string relative to `baseDir`.

fsAudio [String or Array | `[]`] Array of {src: 'url', type: 'MIMEtype'} objects for audio to play upon pausing study due to leaving fullscreen mode, OR string relative to `baseDir`.

startWithA [Boolean | `true`] Whether to start with the 'A' image list on both left and right. If true, both sides start with their respective A image lists; if false, both lists start with their respective B image lists.

randomizeImageOrder [Boolean | `true`] Whether to randomize image presentation order within the lists `leftImagesA`, `leftImagesB`, `rightImagesA`, and `rightImagesB`. If true (default), the order of presentation is randomized. Each time all the images in one list have been presented, the order is randomized again for the next 'round.' If false, the order of presentation is as written in the list. Once all images are presented, we loop back around to the first image and start again.

Example of randomization: suppose we have defined

```
leftImagesA: ['apple', 'banana', 'cucumber'],
leftImagesB: ['aardvark', 'bat'],
randomizeImageOrder: true,
startWithA: true
```

And suppose the timing is such that we end up with 10 images total. Here is a possible sequence of images shown on the left:

```
['banana', 'aardvark', 'apple', 'bat', 'cucumber', 'bat', 'cucumber',
'aardvark', 'apple', 'bat']
```

displayMs [Number | `750`] Amount of time to display each image, in milliseconds

blankMs [Number | `250`] Amount of time for blank display between each image, in milliseconds

border [String | `thin solid gray`] Format of border to display around alternation streams, if any. See https://developer.mozilla.org/en-US/docs/Web/CSS/border for syntax.

backgroundColor [String | `'white'`] Color of background. See CSS specs for acceptable syntax: can use color names ('blue', 'red', 'green', etc.), or rgb hex values (e.g. '#800080' - include the '#')

containerColor [String | `'white'`] Color of image stream container, if different from overall background. Defaults to backgroundColor if one is provided. See CSS specs for acceptable syntax: can use color names ('blue', 'red', 'green', etc.), or rgb hex values (e.g. '#800080' - include the '#')

leftImagesA [Array | `[]`] Set A of images to display on left of screen. Left stream will alternate between images from set A and from set B. Elements of list can be full URLs or relative paths starting from *baseDir*.

leftImagesB [Array | `[]`] Set B of images to display on left of screen. Left stream will alternate between images from set A and from set B. Elements of list can be full URLs or relative paths starting from *baseDir*.

rightImagesA [Array | `[]`] Set A of images to display on right of screen. Right stream will alternate between images from set A and from set B. Elements of list can be full URLs or relative paths starting from *baseDir*.

rightImagesB [Array | `[]`] Set B of images to display on right of screen. Right stream will alternate between images from set A and from set B. Elements of list can be full URLs or relative paths starting from *baseDir*.

3.3.4 Data collected

The fields added specifically for this frame type are:

leftSequence [Array] Sequence of images shown on the left

rightSequence [Array] Sequence of images shown on the right

hasBeenPaused [Boolean] Whether the trial was paused at any point

3.3.5 Events recorded

The events recorded specifically by this frame are:

stoppingCapture Just before stopping webcam video capture

startIntro Immediately before starting intro/announcement segment

startTestTrial Immediately before starting test trial segment

clearImages Records each time images are cleared from display

presentImages Immediately after making images visible

left url of left image

right url of right image

3.4 exp-lookit-exit-survey

3.4.1 Overview

Standard exit survey for Lookit studies. On the first screen, the parent confirms participant birthdate, provides video sharing permission level & Databrary sharing selection, has an option to withdraw video, and can give freeform comments. On the second screen your custom debriefing text is shown.

What it looks like

Recording

There is no recording on this frame unless you set up a session-level recording.

Display

By default, this frame is not shown fullscreen.

More general functionality

Below is information specific to this particular frame. There may also be available parameters, events recorded, and data collected that come from the following more general sources:

- the *base frame* (things all frames do)

3.4.2 Example

This frame will display the exit survey and a debriefing page about the Lookit physics study.

```
"my-exit-survey": {
    "kind": "exp-lookit-exit-survey",
    "debriefing": {
        "title": "Thank you!",
        "text": "PLACEHOLDER DEBRIEFING TEXT. SEE DOCS FOR ADVICE ON WHAT TO INCLUDE␣
↪HERE.",
        "image": {
            "src": "https://s3.amazonaws.com/lookitcontents/ducks/duck.png",
            "alt": "Very cute duck"
        }
    }
}
```

For information about what belongs in the debriefing, see the Lookit docs.

3.4.3 Parameters

debriefing [Object] Object specifying information to show on second page of exit survey, before returning to main Lookit site.

> **title** [String] Title of debriefing page
>
> **text** [String] Debriefing text. You can include line breaks (n) and links (Link text)
>
> **image** [Object] Image to show at top of debriefing (optional)
>
> > **src** [String] Image URL
> >
> > **alt** [String] Image alt-text

showShareButton [Boolean | `true`] Whether to show a 'share this study on Facebook' button

3.4.4 Data collected

The fields added specifically for this frame type are:

birthDate [String] Child's birthdate as entered into exit survey; timestamp string starting YYYY-mm-dd. @attribute birthDate

databraryShare [String] Whether data can be shared with Databrary: 'yes' or 'no' @attribute databraryShare

useOfMedia [String] Video privacy level: 'private', 'scientific', or 'public'

withdrawal [Boolean] Whether the the box to withdraw video data is checked

feedback [String] Freeform comments entered by parent

3.4.5 Events recorded

No events are recorded specifically by this frame.

3.5 exp-lookit-images-audio

3.5.1 Overview

Frame to display image(s) and play audio, with optional video recording. Options allow customization for looking time, storybook, forced choice, and reaction time type trials, including training versions where children (or parents) get feedback about their responses.

This can be used in a variety of ways - for example:

- Display an image for a set amount of time and measure looking time

- Display two images for a set amount of time and play audio for a looking-while-listening paradigm

- Show a "storybook page" where you show images and play audio, having the parent/child press 'Next' to proceed. If desired, images can appear and be highlighted at specific times relative to audio. E.g., the audio might say "This [image of Remy appears] is a boy named Remy. Remy has a little sister [image of Zenna appears] named Zenna. [Remy highlighted] Remy's favorite food is brussel sprouts, but [Zenna highlighted] Zenna's favorite food is ice cream. [Remy and Zenna both highlighted] Remy and Zenna both love tacos!"

- Play audio asking the child to choose between two images by pointing or answering verbally. Show text for the parent about how to help and when to press Next.

- Play audio asking the child to choose between two images, and require one of those images to be clicked to proceed (see "choiceRequired" option).

- Measure reaction time as the child is asked to choose a particular option on each trial (e.g., a central cue image is shown first, then two options at a short delay; the child clicks on the one that matches the cue in some way)

- Provide audio and/or text feedback on the child's (or parent's) choice before proceeding, either just to make the study a bit more interactive ("Great job, you chose the color BLUE!") or for initial training/familiarization to make sure they understand the task. Some images can be marked as the "correct" answer and a correct answer required to proceed. If you'd like to include some initial training questions before your test questions, this is a great way to do it.

In general, the images are displayed in a designated region of the screen with aspect ratio 7:4 (1.75 times as wide as it is tall) to standardize display as much as possible across different monitors. If you want to display things truly fullscreen, you can use `autoProceed` and not provide `parentText` so there's nothing at the bottom, and then set `maximizeDisplay` to true.

Any number of images may be placed on the screen, and their position specified. (Aspect ratio will be the same as the original image.)
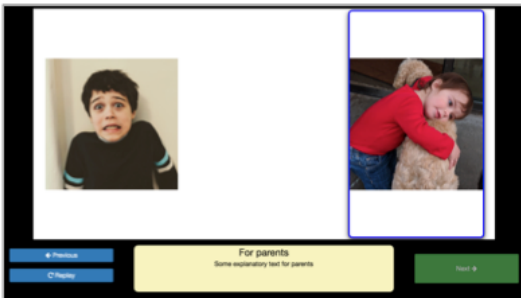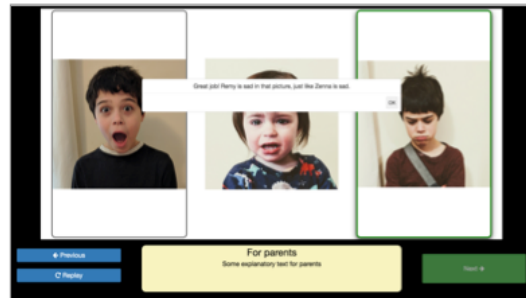
What it looks like
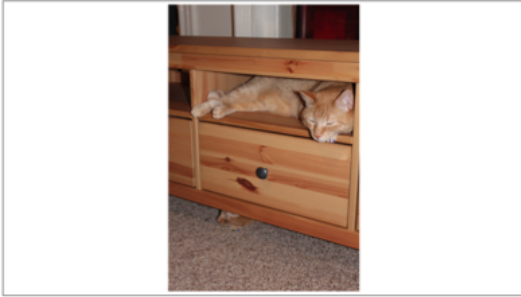
Show image(s) for a set
duration with "maximizeDisplay": true

Display a progress bar

Highlight images at specified times
during audio

Ask children to click one of the images,
and provide feedback (audio and/or text)
on answers

Recording

Webcam recording may be turned on or off; if on, stimuli are not displayed and audio is not started until recording begins. (Using the frame-specific `doRecording` property is good if you have a smallish number of test trials and prefer to have separate video clips for each. For reaction time trials or many short trials, you will likely want to use session recording instead - i.e. start the session recording before the first trial and end after the last trial - to avoid the short delays related to starting/stopping the video.)

Display

This frame is displayed fullscreen, but is not paused or otherwise disabled if the user leaves fullscreen. A button appears prompting the user to return to fullscreen mode.

If the frame before it is not fullscreen, that frame needs to include a manual "next" button so that there's a user interaction event to trigger fullscreen mode. (Browsers don't allow us to switch to FS without a user event.)

Specifying where files are

Several of the parameters for this frame can be specified either by providing a list of full URLs and file types, or by providing just a filename that will be interpreted relative to the baseDir. See the *expand-assets mixin* tool that this frame uses.

More general functionality

Below is information specific to this particular frame. There may also be available parameters, events recorded, and data collected that come from the following more general sources:

- the *base frame* (things all frames do)
- *video-record mixin*
- *expand-assets mixin*

3.5.2 Examples

The examples below show a variety of usages.

1. Single image displayed full-screen, maximizing area on monitor, for 8 seconds.

```
"image-1": {
    "kind": "exp-lookit-images-audio",
    "images": [
        {
            "id": "cats",
            "src": "two_cats.png",
            "position": "fill"
        }
    ],
    "baseDir": "https://www.mit.edu/~kimscott/placeholderstimuli/",
    "autoProceed": true,
    "doRecording": true,
    "durationSeconds": 8,
    "maximizeDisplay": true
}
```

2. Single image displayed at specified position, with 'next' button to move on

```
"image-2": {
    "kind": "exp-lookit-images-audio",
    "images": [
        {
            "id": "cats",
            "src": "three_cats.JPG",
            "top": 10,
            "left": 30,
            "width": 40
        }
    ],
    "baseDir": "https://www.mit.edu/~kimscott/placeholderstimuli/",
    "autoProceed": false,
    "doRecording": true,
    "parentTextBlock": {
```

```
            "text": "Some explanatory text for parents",
            "title": "For parents"
        }
    }
}
```

3. Image plus audio, auto-proceeding after audio completes and 4 seconds go by

```
"image-3": {
    "kind": "exp-lookit-images-audio",
    "audio": "wheresremy",
    "images": [
        {
            "id": "remy",
            "src": "wheres_remy.jpg",
            "position": "fill"
        }
    ],
    "baseDir": "https://www.mit.edu/~kimscott/placeholderstimuli/",
    "audioTypes": [
        "mp3",
        "ogg"
    ],
    "autoProceed": true,
    "doRecording": false,
    "durationSeconds": 4,
    "parentTextBlock": {
        "text": "Some explanatory text for parents",
        "title": "For parents"
    },
    "showProgressBar": true
}
```

4. Image plus audio, with 'next' button to move on

```
"image-4": {
    "kind": "exp-lookit-images-audio",
    "audio": "peekaboo",
    "images": [
        {
            "id": "remy",
            "src": "peekaboo_remy.jpg",
            "position": "fill"
        }
    ],
    "baseDir": "https://www.mit.edu/~kimscott/placeholderstimuli/",
    "audioTypes": [
        "mp3",
        "ogg"
    ],
    "autoProceed": false,
    "doRecording": false,
    "parentTextBlock": {
        "text": "Some explanatory text for parents",
        "title": "For parents"
    }
}
```

5. Two images plus audio question asking child to point to one of the images, demonstrating different timing of image display & highlighting of images during audio

```
"image-5": {
    "kind": "exp-lookit-images-audio",
    "audio": "remyzennaintro",
    "images": [
        {
            "id": "remy",
            "src": "scared_remy.jpg",
            "position": "left"
        },
        {
            "id": "zenna",
            "src": "love_zenna.jpg",
            "position": "right",
            "displayDelayMs": 1500
        }
    ],
    "baseDir": "https://www.mit.edu/~kimscott/placeholderstimuli/",
    "highlights": [
        {
            "range": [
                0,
                1.5
            ],
            "imageId": "remy"
        },
        {
            "range": [
                1.5,
                3
            ],
            "imageId": "zenna"
        }
    ],
    "autoProceed": false,
    "doRecording": true,
    "parentTextBlock": {
        "text": "Some explanatory text for parents",
        "title": "For parents"
    }
}
```

6. Three images with audio prompt, family has to click one of two to continue

```
"image-6": {
    "kind": "exp-lookit-images-audio",
    "audio": "matchremy",
    "images": [
        {
            "id": "cue",
            "src": "happy_remy.jpg",
            "position": "center",
            "nonChoiceOption": true
        },
        {
            "id": "option1",
```

```
            "src": "happy_zenna.jpg",
            "position": "left",
            "displayDelayMs": 2000
        },
        {
            "id": "option2",
            "src": "annoyed_zenna.jpg",
            "position": "right",
            "displayDelayMs": 2000
        }
    ],
    "baseDir": "https://www.mit.edu/~kimscott/placeholderstimuli/",
    "autoProceed": false,
    "doRecording": true,
    "choiceRequired": true,
    "parentTextBlock": {
        "text": "Some explanatory text for parents",
        "title": "For parents"
    },
    "canMakeChoiceBeforeAudioFinished": true
}
```

7. Three images with audio prompt, family has to click correct one to continue - audio feedback on incorrect answer

```
"image-7": {
    "kind": "exp-lookit-images-audio",
    "audio": "matchzenna",
    "images": [
        {
            "id": "cue",
            "src": "sad_zenna.jpg",
            "position": "center",
            "nonChoiceOption": true
        },
        {
            "id": "option1",
            "src": "surprised_remy.jpg",
            "position": "left",
            "feedbackAudio": "negativefeedback",
            "displayDelayMs": 3500
        },
        {
            "id": "option2",
            "src": "sad_remy.jpg",
            "correct": true,
            "position": "right",
            "displayDelayMs": 3500
        }
    ],
    "baseDir": "https://www.mit.edu/~kimscott/placeholderstimuli/",
    "autoProceed": false,
    "doRecording": true,
    "choiceRequired": true,
    "parentTextBlock": {
        "text": "Some explanatory text for parents",
```

```
        "title": "For parents"
    },
    "correctChoiceRequired": true,
    "canMakeChoiceBeforeAudioFinished": false
}
```

8. Three images with audio prompt, family has to click correct one to continue - text feedback on incorrect answer

```
"image-8": {
    "kind": "exp-lookit-images-audio",
    "audio": "matchzenna",
    "images": [
        {
            "id": "cue",
            "src": "sad_zenna.jpg",
            "position": "center",
            "nonChoiceOption": true
        },
        {
            "id": "option1",
            "src": "surprised_remy.jpg",
            "position": "left",
            "feedbackText": "Try again! Remy looks surprised in that␣
→picture. Can you find the picture where he looks sad, like Zenna?",
            "displayDelayMs": 3500
        },
        {
            "id": "option2",
            "src": "sad_remy.jpg",
            "correct": true,
            "position": "right",
            "feedbackText": "Great job! Remy is sad in that picture, just␣
→like Zenna is sad.",
            "displayDelayMs": 3500
        }
    ],
    "baseDir": "https://www.mit.edu/~kimscott/placeholderstimuli/",
    "autoProceed": false,
    "doRecording": true,
    "choiceRequired": true,
    "parentTextBlock": {
        "text": "Some explanatory text for parents",
        "title": "For parents"
    },
    "correctChoiceRequired": true,
    "canMakeChoiceBeforeAudioFinished": false
}
```

3.5.3 Parameters

doRecording [Boolean] Whether to do webcam recording (will wait for webcam connection before starting audio or showing images if so)

autoProceed [Boolean | `false`]

> Whether to proceed automatically when all conditions are met, vs. enabling next button at that point. If true: the next, previous, and replay buttons are hidden, and the frame auto-advances after ALL of the following happen
>
> > (a) the audio segment (if any) completes
> >
> > (b) the durationSeconds (if any) is achieved
> >
> > (c) a choice is made (if required)
> >
> > (d) that choice is correct (if required)
> >
> > (e) the choice audio (if any) completes
> >
> > (f) the choice text (if any) is dismissed
>
> If false: the next, previous, and replay buttons (as applicable) are displayed. It becomes possible to press 'next' only once the conditions above are met.

durationSeconds [Number | `0`] Minimum duration of frame in seconds. If set, then it will only be possible to proceed to the next frame after both the audio completes AND this duration is acheived.

showProgressBar [Boolean | `false`] [Only used if durationSeconds set] Whether to show a progress bar based on durationSeconds in the parent text area.

showPreviousButton [Boolean | `true`] [Only used if not autoProceed] Whether to show a previous button to allow the participant to go to the previous frame

showReplayButton [Boolean | `true`] [Only used if not autoProceed AND if there is audio] Whether to show a replay button to allow the participant to replay the audio

maximizeDisplay [Boolean | `false`] Whether to have the image display area take up the whole screen if possible. This will only apply if (a) there is no parent text and (b) there are no control buttons (next, previous, replay) because the frame auto-proceeds.

audio [String or Array | `[]`] Audio file to play at the start of this frame. This can either be an array of {src: 'url', type: 'MIMEtype'} objects, e.g. listing equivalent .mp3 and .ogg files, or can be a single string *filename* which will be expanded based on *baseDir* and *audioTypes* values (see *audioTypes*).

parentTextBlock [Object | `{}`] Text block to display to parent. (Each field is optional)

> > **title** title to display
> >
> > **text** text paragraph of text
> >
> > **css** object specifying any css properties to apply to this section, and their values - e.g. `{'color': 'gray', 'font-size':  'large'}`

images [Array | `[]`] Array of images to display and information about their placement. For each image, you need to specify `src` (image name/URL) and placement (either by providing left/width/top values, or by using a `position` preset). Everything else is optional! This is where you would say that an image should be shown at a delay, or specify times to highlight particular images.

> > **id** [String] unique ID for this image. This must not have any spaces or special characters and cannot start with a number.
> >
> > **src** [String] URL of image source. This can be a full URL, or relative to baseDir (see baseDir).

---

alt [String] alt-text for image in case it doesn't load and for screen readers

left [Number] left margin, as percentage of story area width. If not provided, the image is centered horizontally.

width [Number] image width, as percentage of story area width. Note: in general only provide one of width and height; the other will be adjusted to preserve the image aspect ratio.

top [Number] top margin, as percentage of story area height. If not provided, the image is centered vertically.

height [Number] image height, as percentage of story area height. Note: in general only provide one of width and height; the other will be adjusted to preserve the image aspect ratio.

position [String] one of 'left', 'center', 'right', 'fill' to use presets that place the image in approximately the left, center, or right third of the screen or to fill the screen as much as possible. This overrides left/width/top values if given.

nonChoiceOption [Boolean] [Only used if `choiceRequired` is true] whether this should be treated as a non-clickable option (e.g., this is a picture of a girl, and the child needs to choose whether the girl has a DOG or a CAT)

displayDelayMs [Number] Delay at which to show the image after trial start (timing will be relative to any audio or to start of trial if no audio). Optional; default is to show images immediately.

feedbackAudio [Array or String] [Only used if `choiceRequired` is true] Audio to play upon clicking this image. This can either be an array of {src: 'url', type: 'MIMEtype'} objects, e.g. listing equivalent .mp3 and .ogg files, or can be a single string `filename` which will be expanded based on `baseDir` and `audioTypes` values (see `audioTypes`).

feedbackText [String] [Only used if `choiceRequired` is true] Text to display in a dialogue window upon clicking the image.

backgroundColor [String | `'black'`] Color of background. See CSS specs for acceptable syntax: can use color names ('blue', 'red', 'green', etc.), or rgb hex values (e.g. '#800080' - include the '#')

pageColor [String | `'white'`] Color of area where images are shown, if different from overall background. Defaults to backgroundColor if one is provided. See CSS specs for acceptable syntax: can use color names ('blue', 'red', 'green', etc.), or rgb hex values (e.g. '#800080' - include the '#')

choiceRequired [Boolean | `false`] Whether this is a frame where the user needs to click to select one of the images before proceeding.

correctChoiceRequired [Boolean | `false`] [Only used if *choiceRequired* is true] Whether the participant has to select one of the *correct* images before proceeding.

canMakeChoiceBeforeAudioFinished [Boolean | `false`] Whether the participant can make a choice before audio finishes. (Only relevant if *choiceRequired* is true.)

highlights [Array | `[]`] Array representing times when particular images should be highlighted. Each element of the array should be of the form {'range': [3.64, 7.83], 'imageId': 'myImageId'}. The two *range* values are the start and end times of the highlight in seconds, relative to the audio played. The *imageId* corresponds to the *id* of an element of *images*. Highlights can overlap in time. Any that go longer than the audio will just be ignored/cut off. One strategy for generating a bunch of highlights for a longer story is to annotate using Audacity and export the labels to get the range values.

range [Array] [startTimeInSeconds, endTimeInSeconds], e.g. [3.64, 7.83]

imageId [String] ID of the image to highlight, corresponding to the id field of the element of images to highlight

3.5.4 Data collected

The fields added specifically for this frame type are:

images [Array] Array of images used in this frame [same as passed to this frame, but may reflect random assignment for this particular participant]

selectedImage [String] ID of image selected at time of proceeding

correctImageSelected [Boolean] Whether image selected at time of proceeding is marked as correct

audioPlayed [String] Source URL of audio played, if any. If multiple sources provided (e.g. mp4 and ogg versions) just the first is stored.

3.5.5 Events recorded

The events recorded specifically by this frame are:

videoStarted When video begins playing (recorded each time video starts if played through more than once)

replayAudio When main audio segment is replayed

trialComplete Trial is complete and attempting to move to next frame; may wait for recording to catch up before proceeding.

finishAudio When main audio segment finishes playing

startTimer Timer for set-duration trial begins

endTimer Timer for set-duration trial ends

startAudio When main audio segment starts playing

failedToStartAudio When main audio cannot be started. In this case we treat it as if the audio was completed (for purposes of allowing participant to proceed)

displayAllImages When images are displayed to participant (for images without any delay added)

displayImage When a specific image is shown at a delay.

> **imageId** [String] ID of image shown

clickImage When one of the image options is clicked during a choice frame

> **imageId** [String] ID of the image selected

> **correct** [Boolean] whether this image is marked as correct

startImageAudio When image/feedback audio is started

> **imageId** [String] ID of the associated image

failedToStartImageAudio When image/feedback audio cannot be started. In this case we treat it as if the audio was completed (for purposes of allowing participant to proceed)

> **imageId** [String] ID of the associated image

dismissFeedback When the participant dismisses a feedback dialogue

> **imageId** [String] ID of the associated image

3.6 exp-lookit-images-audio-infant-control

3.6.1 Overview

Infant-controlled version of the *exp-lookit-images-audio* frame. This works the same way as `exp-lookit-images-audio` except that you can enable the parent to:

- end the trial by pressing the `endTrialKey` key

- hold down the `lookawayKey` (or the mouse button) to indicate that the child is not looking; the trial will automatically end after the lookaway criterion is met. If the 'lookawayTone' is not 'none' a noise is played while the child is looking away to help the parent know the looking coding is working.

You can disable either of these behaviors by setting the corresponding key to `''`.

The frame will still end when it would have anyway if neither of these things happen! For instance, if you would have displayed an image for 30 seconds, then after 30 seconds the frame will move on, serving as a "ceiling" on looking time.

Lookaway criterion

You have two options for how to determine when the child has looked away long enough to proceed.

1. Set the `lookawayType` to `"total"` to accumulate lookaway time until the child has looked away for a total of *lookawayThreshold* seconds. (For instance, if the `lookawayThreshold` is 2, then the trial will end after the child looks away for 0.5s, then 1s, then 0.5s.) 2. Set the `lookawayType` to `"continuous"` to require that the child look away for a continuous `lookawayThreshold`-second interval. (For instance, if the `lookawayThreshold` is 2, then the child might look away for 1s, 1.5s, and 1s but the trial would continue until she looked away for 2s.)

When looking time is measured

The looking time measurement begins only when the video starts, not while a video connection is established.

If a `lookawayKey` is defined, lookaways are recorded the entire time the frame is running. However, the looking time measurement only starts once images are displayed (or the "delay" timer starts counting down, for images shown at a delay - but e.g., not during webcam connection). Lookaways at the very start don't count! If the child is not looking at the start, the measurement begins once they look for the first time.

3.6.2 Examples

This frame will display an image of some cats for up to 30 seconds. Once the child looks away for more than 2 s total, as coded by the parent holding down the P key, it will proceed.

```
"looking-time-ceil-30s": {
    "kind": "exp-lookit-images-audio-infant-control",
    "lookawayKey": "p",
    "lookawayType": "total",
    "lookawayThreshold": 2,
    "endTrialKey": "q",
    "lookawayTone": "noise",
    "lookawayToneVolume": 0.25,

    "images": [
        {
```

(continues on next page)

```
            "id": "cats",
            "src": "two_cats.png",
            "position": "fill"
        }
    ],
    "baseDir": "https://www.mit.edu/~kimscott/placeholderstimuli/",
    "autoProceed": true,
    "doRecording": true,
    "durationSeconds": 30,
    "maximizeDisplay": true
}
```

3.6.3 Parameters

The parameters for this frame are the same as for *exp-lookit-images-audio*, plus the additional parameters provided by the *infant-controlled-timing mixin*.

3.6.4 Data collected

This frame collects the same data as *exp-lookit-images-audio*, plus the additional data provided by the *infant-controlled-timing mixin*.

3.6.5 Events recorded

This frame records the same events as *exp-lookit-images-audio*, plus the additional events recorded by the *infant-controlled-timing mixin*.

3.7 exp-lookit-instruction-video

3.7.1 Overview

A frame to display video instructions to the user.

A video is displayed to the left, and a transcript or summary in a scrolling box to the right. (The transcript can be omitted if desired, but in that case you must provide complete captioning for the video!)

The participant is required to either scroll to the bottom of the transcript or watch the video to proceed.

What it looks like
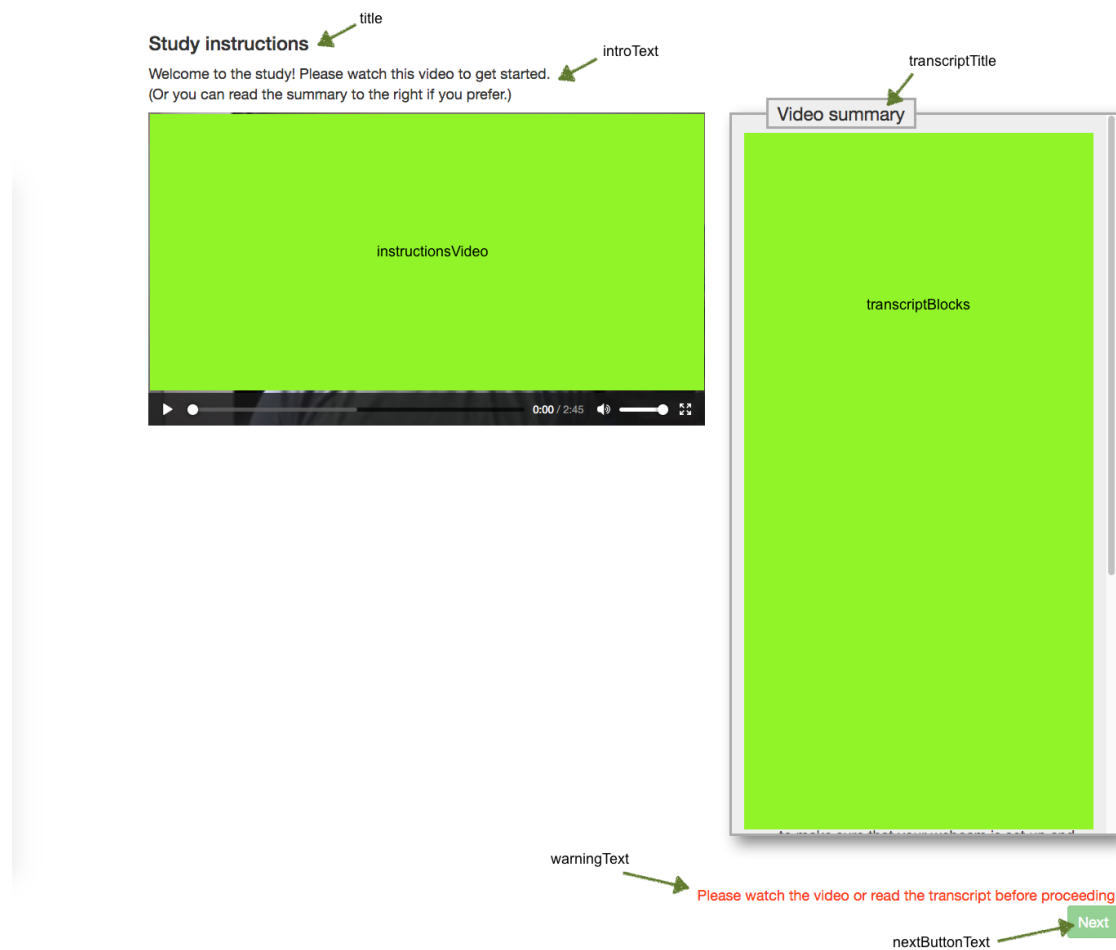
Specifying where files are

Several of the parameters for this frame can be specified either by providing a list of full URLs and file types, or by providing just a filename that will be interpreted relative to the baseDir. See the *expand-assets mixin* tool that this frame uses.

More general functionality

Below is information specific to this particular frame. There may also be available parameters, events recorded, and data collected that come from the following more general sources:

- the *base frame* (things all frames do)

- *expand-assets mixin*

3.7.2 Examples

This frame will show an instruction video & summary.

```
"intro-video": {
    "kind": "exp-lookit-instruction-video",
    "instructionsVideo": [
        {
            "src": "https://raw.github.com/UCDOakeslab/Baby_MR_Lookit/master/Lookit␣
→Introduction Part 1_Edited.mp4",
            "type": "video/mp4"
        }
    ],
    "introText": "Welcome to the study! Please watch this video to get started. \n(Or␣
→you can read the summary to the right if you prefer.)",
    "transcriptTitle": "Video summary",
    "transcriptBlocks": [
        {
            "title": "Background information about the study",
            "listblocks": [
                {
                    "text": "Your baby does not need to be with you at this point in␣
→the study. We will let you know when it is time to get your baby."
                },
                {
                    "text": "Mental rotation, or the ability to manipulate internal␣
→representations of objects, is an important spatial ability. Spatial abilities are␣
→important for understanding objects, reading maps, mathematical reasoning, and␣
→navigating the world. Thus, the development of mental rotation is an important␣
→milestone. In the current study, we are interested in examining whether babies in␣
→general can mentally rotate simple objects."
                }
            ]
        },
        {
            "title": "Preview of what your baby will see"
        },
        {
            "listblocks": [
                {
                    "text": "Your baby will be shown two identical Tetris shapes on␣
→the screen; one on the left and one on the right. The shapes appear and disappear,␣
→changing their orientation each time they reappear. On one side, the rotation will␣
→always be possible. Sometimes, on the other side, a mirror image of the shape will␣
→be presented. If babies can mentally rotate objects, they should spend different␣
→amounts of time watching these two kinds of stimuli."
                }
            ]
        },
        {
            "title": "What's next?",
            "listblocks": [
                {
                    "text": "Because this is an online study, we will check to make␣
→sure that your webcam is set up and working properly on the next page, so we can␣
→record your baby's looking behavior during the study."
                },
                {
```

```
                "text": "Following that page, you will be given an opportunity to␣
→review the consent information and we will ask that you record a short video of␣
→yourself giving consent to participate in this study."
            },
            {
                "text": "We will then ask you questions about your baby's motor␣
→abilities."
            },
            {
                "text": "After you are finished with the consent page and␣
→questions, you will be provided with more detailed information about what to do␣
→during the study and how to get started."
            }
        ]
    }
],
"warningText": "Please watch the video or read the transcript before proceeding.",
"nextButtonText": "Next",
"title": "Study instructions",
"showPreviousButton": false
}
```

3.7.3 Parameters

title [String] Title to show at top of frame

introText [String | 'Welcome! Please watch this video to learn how the study will work. You can r
Intro text to show at top of frame

warningText [String | 'Please watch the video or read the transcript before proceeding.']
Text to show above Next button if participant has not yet watched video or read transcript

instructionsVideo [String or Array] The location of the instructions video to play. This can be either an array of
{'src': 'https://…', 'type': '…'} objects (e.g. providing both webm and mp4 versions at specified URLS) or a
single string relative to baseDir/<EXT>/.

transcriptTitle [String | 'Video transcript' Title to show above video transcript/overview. Generally this
should be either "Video transcript" or "Video summary" depending on whether you're providing a word-for-
word transcript or a condensed summary.

transcriptBlocks [Array] Array of blocks for *exp-text-block*, providing a transcript of the video or an overview of
what it said. A transcript can be broken down into bullet points to make it more readable.

If you've also provided closed captions throughout the video, you can use this space just to provide key points.

If this is left blank ([]) no transcript is displayed.

Each block may have…

> **title** [String] Title of this section
>
> **text** [String] Text of this section
>
> **listblocks** [Array] Bullet points for this section. Each bullet may have…
>
> > **text** Text of this bullet point
> >
> > **image** [Object] Image for this bullet point, with fields:
> >
> > > **src** [String] URL of image

> **alt** [String] Alt text for image

showPreviousButton [Boolean | `true`] Whether to show a 'previous' button

nextButtonText [String | `'Start the videos! \n (You\'ll have a moment to turn around.)'`] Text to display on the 'next frame' button

3.7.4 Data collected

No additional data is collected specifically by this frame type.

3.7.5 Events recorded

No events are recorded specifically by this frame.

3.8 exp-lookit-instructions

3.8.1 Overview

A frame to display instructions to the user. The user's webcam may optionally be displayed, and audio and video clips may be included in the instructions (and may be required to be played before moving on).

What it looks like

More general functionality

Below is information specific to this particular frame. There may also be available parameters, events recorded, and data collected that come from the following more general sources:

- the *base frame* (things all frames do)

3.8.2 Examples

```
"instructions": {
    "kind": "exp-lookit-instructions",
    "blocks": [
        {
            "title": "At vero eos",
            "listblocks": [
                {
                    "text": "At vero eos et accusamus et iusto odio dignissimos␣
→ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores␣
→et quas molestias excepturi sint occaecati cupiditate non provident, similique sunt␣
→in culpa qui officia deserunt mollitia animi, id est laborum et dolorum fuga."
                },
                {
                    "text": "Et harum quidem rerum facilis est et expedita distinctio.
→"
                }
            ]
        },
        {
            "text": "Ut enim ad minim veniam, quis nostrud exercitation ullamco␣
→laboris nisi ut aliquip ex ea commodo consequat.",
            "image": {
                "alt": "Father holding child looking over his shoulder",
                "src": "https://s3.amazonaws.com/lookitcontents/exp-physics/
→OverShoulder.jpg"
            },
            "title": "Lorem ipsum"
        },
        {
            "text": "unde omnis iste natus error sit voluptatem accusantium␣
→doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore␣
→veritatis et quasi architecto beatae vitae dicta sunt explicabo.",
            "title": "Sed ut perspiciatis",
            "mediaBlock": {
                "text": "You should hear 'Ready to go?'",
                "isVideo": false,
                "sources": [
                    {
                        "src": "https://s3.amazonaws.com/lookitcontents/exp-physics-
→final/audio/ready.mp3",
                        "type": "audio/mp3"
                    },
                    {
                        "src": "https://s3.amazonaws.com/lookitcontents/exp-physics-
→final/audio/ready.ogg",
                        "type": "audio/ogg"
                    }
```

(continues on next page)

```
                    ],
                    "mustPlay": true,
                    "warningText": "Please try playing the sample audio."
                }
            },
            {
                "text": "quia voluptas sit aspernatur aut odit aut fugit, sed quia␣
→consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt.",
                "title": "Nemo enim ipsam voluptatem",
                "mediaBlock": {
                    "text": "Look at that.",
                    "isVideo": true,
                    "sources": [
                        {
                            "src": "https://s3.amazonaws.com/lookitcontents/exp-physics-
→final/examples/7_control_same.mp4",
                            "type": "video/mp4"
                        },
                        {
                            "src": "https://s3.amazonaws.com/lookitcontents/exp-physics-
→final/examples/7_control_same.webm",
                            "type": "video/webm"
                        }
                    ],
                    "mustPlay": false
                }
            }
        ],
        "showWebcam": true,
        "webcamBlocks": [
            {
                "title": "Neque porro quisquam",
                "listblocks": [
                    {
                        "text": "Ut enim ad minima veniam, quis nostrum exercitationem␣
→ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur?"
                    },
                    {
                        "text": "Quis autem vel eum iure reprehenderit qui in ea␣
→voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum␣
→fugiat quo voluptas nulla pariatur?"
                    }
                ]
            }
        ],
        "nextButtonText": "Next"
}
```

3.8.3 Parameters

showWebcam [Boolean | `false`] Whether to display the user's webcam

blocks [Array] Array of blocks to be rendered by *exp-text-block*, specifying text/images of instructions to display. In addition to the standard options allowed by `exp-text-block` (text, title, etc.) these blocks may have a field `mediaBlock` with fields:

> **title** [String] Title of section
>
> **text** [String] Text displayed below title
>
> **warningText** [String] Warning text shown if `mustPlay` is true and user moves on without playing media
>
> **sources** [Array] List of objects indicating where media is located, each with fields:
>
> > **src** [String] URL of media file
> >
> > **type** [String] MIMEtype of media file, e.g. `'video/mp4'`
>
> **isVideo** [Boolean] Whether this is a video file, vs. audio
>
> **mustPlay** [Boolean] whether to require user to play this to move on

webcamBlocks [Array] Array of objects specifying text/images of instructions to display under webcam view (if webcam is shown), rendered by *exp-text-block*

showPreviousButton [Boolean | `true`] Whether to show a 'previous' button

nextButtonText [String | `'Start the videos! \n (You\'ll have a moment to turn around.)'`] Text to display on the 'next frame' button

3.8.4 Data collected

No data is collected specifically for this frame type.

3.8.5 Events recorded

No events are recorded specifically by this frame.

3.9 exp-lookit-mood-questionnaire

3.9.1 Overview

A simple mood survey with questions about factors that might affect a child's responses. Includes Likert-type ratings of the CHILD's position on the following scales:

- Tired to Rested
- Sick to Healthy
- Fussy to Happy
- Calm to Active

and of the PARENT's position on:

- Tired to Energetic

- Overwhelmed to On top of things

- Upset to Happy

It also asks for a response in hours:minutes for: - how long ago the child last woke up from sleep or a nap - how long until he/she is due for another nap/sleep (if regular nap schedule) - how long ago the child last ate/drank

and for what the child was doing just before this (free-response). Responses to all questions are required to move on.

What it looks like

More general functionality

Below is information specific to this particular frame. There may also be available parameters, events recorded, and data collected that come from the following more general sources:

- the *base frame* (things all frames do)

3.9.2 Example

```
"mood-survey": {
    "introText": "How are you two doing? We really want to know: we're interested in␣
→how your child's mood affects his or her looking preferences.",
    "kind": "exp-lookit-mood-questionnaire"
}
```

3.9.3 Parameters

introText [String] Intro paragraph describing why we want mood info

3.9.4 Data collected

The fields added specifically for this frame type are:

rested [String] Rating for CHILD on tired - rested scale, '1' to '7' where '7' is rested

healthy [String] Rating for CHILD on sick - healthy scale, '1' to '7' where '7' is healthy

childHappy [String] Rating for CHILD on fussy - happy scale, '1' to '7' where '7' is happy

active [String] Rating for CHILD on calm - active scale, '1' to '7' where '7' is active

energetic [String] Rating for PARENT on tired - energetic scale, '1' to '7' where '7' is energetic

ontopofstuff [String] Rating for PARENT on overwhelmed - on top of stuff scale, '1' to '7' where '7' is on top of stuff

parentHappy [String] Rating for PARENT on upset - happy scale, '1' to '7' where '7' is happy

napWakeUp [String] how long since the child woke up from nap, HH:mm

usualNapSchedule [String] whether the child has a typical nap schedule: 'no', 'yes', or 'yes-overdue' if child is overdue for nap

nextNap [String] only valid if usualNapSchedule is 'yes'; how long until child is due to sleep again, HH:mm

lastEat [String] how long since the child ate/drank, HH:mm

doingBefore [String] what the child was doing before this (free response)

3.9.5 Events recorded

No events are recorded specifically by this frame.

3.10 exp-lookit-observation

3.10.1 Overview

A frame to collect a video observation with the participant's help. By default the webcam is displayed to the participant and they can choose when to start, pause, and resume recording. The duration of an individual recording can optionally be limited and/or recording can be started automatically. This is intended for cases where we want the parent to perform some test or behavior with the child, rather than presenting stimuli ourselves. E.g., you might give instructions to conduct a structured interview and allow the parent to control recording.

What it looks like

Time to do the joke!   Show

    • Rip the paper

    • Wait ten seconds

Webcam feed
not displayed
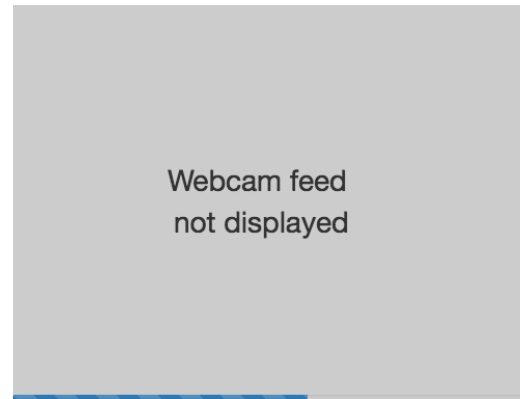
● Recording...   ▌▌ Pause

Previous   Next

More general functionality

Below is information specific to this particular frame. There may also be available parameters, events recorded, and
data collected that come from the following more general sources:

- the *base frame* (things all frames do)
- *video-record mixin*

3.10.2 Examples

This frame will play through a central video of Kim introducing an apple two times, then proceed.

```
"observation": {
    "kind": "exp-lookit-observation",
    "blocks": [
        {
            "title": "Lorem ipsum dolor sit amet",
            "listblocks": [
                {
                    "text": "consectetur adipiscing elit, sed do eiusmod tempor
→incididunt ut labore et dolore magna aliqua."
                },
                {
                    "text": "Ut enim ad minim veniam, quis nostrud exercitation
→ullamco laboris nisi ut aliquip ex ea commodo consequat."
                }
```

```
            ]
        }
    ],
    "hideWebcam": true,
    "hideControls": false,
    "recordSegmentLength": 10,
    "startRecordingAutomatically": false,
    "nextButtonText": "move on",
    "showPreviousButton": false
}
```

3.10.3 Parameters

hideWebcam [Boolean | `false`] Whether to hide webcam view when frame loads (participant will still be able to show manually)

blocks [Array] Array of blocks specifying specifying text/images of instructions to display, rendered by *exp-text-block*

showPreviousButton [Boolean | `true`] Whether to show a 'previous' button

nextButtonText [String | `'Next'`] Text to display on the 'next frame' button

recordSegmentLength [Number | `300`] Number of seconds to record for before automatically pausing. Use 0 for no limit.

startRecordingAutomatically [Boolean | `false`] Whether to automatically begin recording upon frame load

recordingRequired [Number | `0`] Whether a recording must be made to proceed to next frame. 'Next' button will be disabled until recording is made if so. 0 to not require recording; any positive number to require that many seconds of recording

hideControls [Boolean | `false`] Whether to hide video recording controls (only use with startRecordingAutomatically)

3.10.4 Data collected

No fields are added specifically for this frame type.

3.10.5 Events recorded

The events recorded specifically by this frame are:

webcamHidden Webcam display hidden at start of frame due to `hideWebcam` parameter

hideWebcam Webcam display toggled off by participant

showWebcam Webcam display toggled on by participant

recorderTimeout Video recording automatically paused upon reaching time limit

3.11 exp-lookit-start-recording

3.11.1 Overview

Dedicated frame to start session recording.

This frame will take a few seconds to get a session-level recording started, then proceed immediately to the next frame. (See Lookit docs for information about session-level vs. individual-frame recording.)

(You could also set startSessionRecording to true on any frame, but then you need to rely on that individual frame's setup for waiting for recording before getting started.)

Just like for *exp-lookit-calibration*, you can display a video or an optionally animated image (see below for examples of each) as a placeholder while getting recording started.

What it looks like

establishing video connection
please wait...

Display

This can be displayed full-screen or not. If the following frame is full-screen, make this one full-screen too since there will not be a user button press to start the next frame.

Specifying where files are

Several of the parameters for this frame can be specified either by providing a list of full URLs and file types, or by providing just a filename that will be interpreted relative to the baseDir. See the *expand-assets mixin* tool that this frame uses.

More general functionality

Below is information specific to this particular frame. There may also be available parameters, events recorded, and data collected that come from the following more general sources:

- the *base frame* (things all frames do)

- *expand-assets mixin*

3.11.2 Examples

This frame will start a session-level recording, showing a spinning image until the recording starts:

```
"start-recording-with-image": {
    "kind": "exp-lookit-start-recording",
    "baseDir": "https://www.mit.edu/~kimscott/placeholderstimuli/",
    "videoTypes": [
        "webm",
        "mp4"
    ],
    "image": "peekaboo_remy.jpg",
    "imageAnimation": "spin"
    "displayFullscreen": true
}
```

This frame will start a session-level recording, showing a looping video until the recording starts:

```
"start-recording-with-video": {
    "kind": "exp-lookit-start-recording",
    "baseDir": "https://www.mit.edu/~kimscott/placeholderstimuli/",
    "videoTypes": [
        "webm",
        "mp4"
    ],
    "video": "attentiongrabber",
    "displayFullscreen": true
}
```

3.11.3 Parameters

displayFullscreen [Boolean | `true`] Whether to display this frame in full-screen mode

backgroundColor [String | `'white'`] Color of background. See CSS specs for acceptable syntax: can use color names ('blue', 'red', 'green', etc.), or rgb hex values (e.g. '#800080' - include the '#')

video [String or Array] Video to play (looping) while waiting. You can optionally supply either a video or image, not both.

This can be either an array of {src: 'url', type: 'MIMEtype'} objects or just a string like *attentiongrabber* to rely on the *baseDir* and *videoTypes* to generate full paths.

image [String] Image to display while waiting. You can optionally supply either a video or image, not both.

This can be either a full URL or just the filename (e.g. "star.png") to use the full path based on *baseDir* (e.g. *baseDir/img/star.png*).

imageAnimation [String | `'spin'`] Which animation to use for the image. Options are 'bounce', 'spin', or '' (empty to not animate).

3.11.4 Data collected

No fields are added specifically for this frame type.

3.11.5 Events recorded

No events are recorded specifically by this frame.

3.12 exp-lookit-stimuli-preview

3.12.1 Overview

A frame that to explain any blinding procedures to parents, and offer them the option to preview stimuli before the study. Two buttons are available to move on: one turns on video recording & shows the stimuli previews (if the parent wants to preview stimuli), and one moves on to the next frame that (if the parent declines).

What it looks like
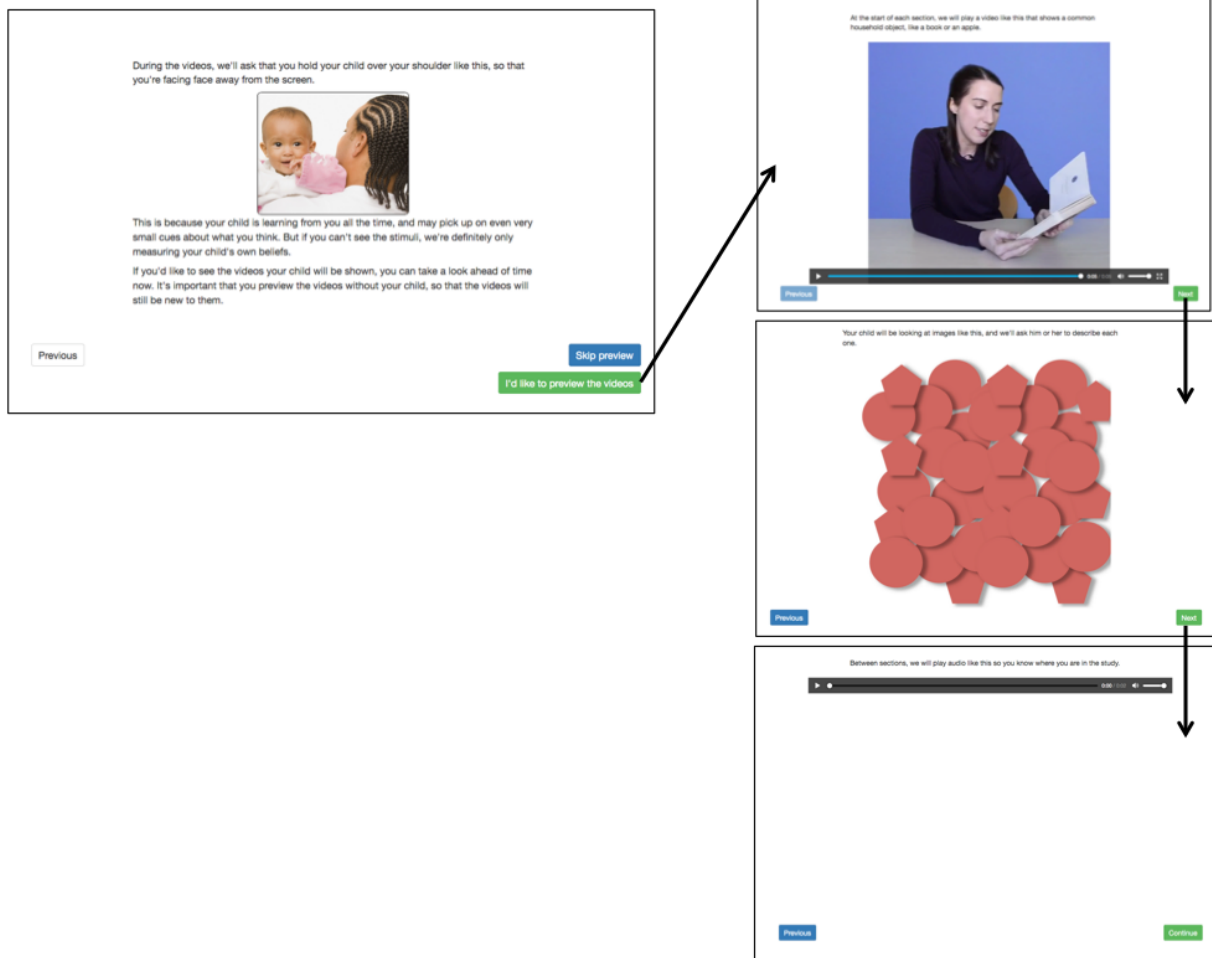
Recording

If `doRecording` is true, then the family is recorded while previewing stimuli if they choose to. This is so that if you want to make sure the child didn't see stimuli ahead of the study you can check.

Specifying where files are

Several of the parameters for this frame can be specified either by providing a list of full URLs and file types, or by providing just a filename that will be interpreted relative to the baseDir. See the *expand-assets mixin* tool that this frame uses.

More general functionality

Below is information specific to this particular frame. There may also be available parameters, events recorded, and data collected that come from the following more general sources:

- the *base frame* (things all frames do)

- *video-record mixin*

- *expand-assets mixin*

3.12.2 Example

```
"my-video-preview": {
    "kind": "exp-lookit-stimuli-preview",
    "doRecording": true,
    "skipButtonText": "Skip preview",
    "previewButtonText": "I'd like to preview the videos",
    "blocks": [
        {
            "text": "During the videos, we'll ask that you hold your child over your
→shoulder like this, so that you're facing face away from the screen."
        },
        {
            "image": {
                "alt": "Father holding child looking over his shoulder",
                "src": "https://raw.githubusercontent.com/kimberscott/placeholder-
→stimuli/master/img/OverShoulder.jpg"
            }
        },
        {
            "text": "This is because your child is learning from you all the time,
→and may pick up on even very small cues about what you think. But if you can't see
→the stimuli, we're definitely only measuring your child's own beliefs."
        },
        {
            "text": "If you'd like to see the videos your child will be shown, you
→can take a look ahead of time now. It's important that you preview the videos
→without your child, so that the videos will still be new to them."
        }
    ],
    "showPreviousButton": true,
    "baseDir": "https://raw.githubusercontent.com/kimberscott/placeholder-stimuli/
→master/",
    "videoTypes": ["webm", "mp4"],
    "audioTypes": ["mp3", "ogg"],
    "stimuli": [
        {
            "caption": "At the start of each section, we will play a video like this
→that shows a common household object, like a book or an apple.",
```

(continues on next page)

```
          "video": "cropped_book"
        },
        {
          "caption": "Your child will be looking at images like this, and we'll ask
→him or her to describe each one.",
          "image": "square.png"
        },
        {
          "caption": "Between sections, we will play audio like this so you know where
→you are in the study.",
          "audio": "sample_1"
        }
    ]
}
```

3.12.3 Parameters

video [Object | `{}`] Object describing the video to show. It can have the following properties:

>   **source** [String or Array] The location of the main video to play. This can be either an array of `{'src': 'https://...', 'type': '...'}` objects (e.g., to provide both webm and mp4 versions at specified URLS) or a single string relative to `baseDir/<EXT>/`.

showPreviousButton [Boolean | `true`] Whether to show a 'previous' button

blocks [Array] Array of text blocks to display as an introduction to the preview. Should be a list of objects that can be passed to *exp-text-block*; each can have any of the properties below.

previewButtonText [String | `'I\'d like to preview the videos'`] Text on the preview button user clicks to proceed to stimuli/images

skipButtonText [String | `'Skip preview'`] Text to display on the button to skip the next frame

stimuli [Array] An array of preview stimuli to display within a single frame, defined as an array of objects. Generally each item of the list will include ONE of image, video, or audio (depending on the stimulus type), plus a caption. Each item can have fields:

>   **caption** [String] Some text to appear above this video/audio/image
>
>   **video** [String or Array] String indicating video URL. This can be relative to baseDir, OR Array of {src: 'url', type: 'MIMEtype'} objects.
>
>   **audio** [String or Array] String indicating audio URL. This can be relative to baseDir, OR Array of {src: 'url', type: 'MIMEtype'} objects.
>
>   **image** [String] URL of image to display. Can be full path or relative to baseDir/img.

nextStimulusText [String | `'Next'`] Text on the button to proceed to the next example video/image

previousStimulusText [String | `'Previous'`] Text on the button to proceed to the previous example video/image

doRecording [Boolean | `true`] Whether to make a webcam video recording during stimulus preview (begins only if user chooses to preview stimuli). Default true.

3.12.4 Data collected

No fields are added specifically for this frame type.

3.12.5 Events recorded

The events recorded specifically by this frame are:

startPreview User clicks on start preview button

nextStimulus User clicks to move to next stimulus

previousStimulus User clicks to move to previous stimulus

3.13 exp-lookit-stop-recording

3.13.1 Overview

Dedicated frame to stop session recording.

This frame will take a few seconds to upload an ongoing session-level recording, then proceed immediately to the next frame. (See Lookit docs for information about session-level vs. individual-frame recording.)

It will time out after a default of 5 minutes of waiting for the upload to complete, or after 5 seconds of not seeing any progress (i.e. something went wrong with starting the upload process). If there is no active session recording, it proceeds immediately.

(You could also set stopSessionRecording to true on any frame, but you generally wouldn't get any specialized functionality for displaying a nice message about upload progress.)

Just like for *exp-lookit-calibration*, you can display a video or an optionally animated image (see below for examples of each) as a placeholder while getting recording started.

What it looks like

Uploading video... 72%
please wait...

Display

This can be displayed full-screen or not. If the following frame is full-screen, make this one full-screen too since there will not be a user button press to start the next frame.

Specifying where files are

Several of the parameters for this frame can be specified either by providing a list of full URLs and file types, or by providing just a filename that will be interpreted relative to the `baseDir`. See the *expand-assets mixin* tool that this frame uses.

More general functionality

Below is information specific to this particular frame. There may also be available parameters, events recorded, and data collected that come from the following more general sources:

- the *base frame* (things all frames do)
- *expand-assets mixin*

3.13.2 Examples

This frame will stop a session-level recording, showing a spinning image until the recording is uploaded:

```
"stop-recording-with-image": {
    "kind": "exp-lookit-stop-recording",
    "baseDir": "https://www.mit.edu/~kimscott/placeholderstimuli/",
    "videoTypes": [
        "webm",
        "mp4"
    ],
    "image": "peekaboo_remy.jpg",
    "imageAnimation": "spin"
    "displayFullscreen": true
}
```

This frame will stop a session-level recording, showing a looping video until the recording is uploaded:

```
"stop-recording-with-video": {
    "kind": "exp-lookit-stop-recording",
    "baseDir": "https://www.mit.edu/~kimscott/placeholderstimuli/",
    "videoTypes": [
        "webm",
        "mp4"
    ],
    "video": "attentiongrabber",
    "displayFullscreen": true
}
```

3.13.3 Parameters

displayFullscreen [Boolean | `true`] Whether to display this frame in full-screen mode

backgroundColor [String | `'white'`] Color of background. See CSS specs for acceptable syntax: can use color names ('blue', 'red', 'green', etc.), or rgb hex values (e.g. '#800080' - include the '#')

video [String or Array] Video to play (looping) while waiting. You can optionally supply either a video or image, not both.

This can be either an array of {src: 'url', type: 'MIMEtype'} objects or just a string like *attentiongrabber* to rely on the *baseDir* and *videoTypes* to generate full paths.

image [String] Image to display while waiting. You can optionally supply either a video or image, not both.

This can be either a full URL or just the filename (e.g. "star.png") to use the full path based on *baseDir* (e.g. *baseDir/img/star.png*).

imageAnimation [String | `'spin'`] Which animation to use for the image. Options are 'bounce', 'spin', or '' (empty to not animate).

sessionMaxUploadSeconds: [Number | `3000`] Maximum time allowed for whole-session video upload before proceeding, in seconds. Can be overridden by researcher, based on tradeoff between making families wait and losing data.

3.13.4 Data collected

No fields are added specifically for this frame type.

3.13.5 Events recorded

The events recorded specifically by this frame are:

warningNoActiveSessionRecording If there's no active session recording so this frame is proceeding immediately.

warningUploadTimeoutError If no progress update about upload is available within 10s, and frame proceeds automatically. Otherwise if the upload has started (e.g. we know it is 10% done) it will continue waiting.

3.14 exp-lookit-survey

3.14.1 Overview

Basic survey frame allowing researcher to specify question text and types.

If a participant returns to this frame after continuing, via a 'Previous' button on the next frame, then the values in this form are pre-filled.

AlpacaJS

This frame uses ember-cli-dynamic-forms as a wrapper for alpacajs, a powerful library for generating online forms. To specify the structure of your form (questions, answer types, validation), you provide a single 'formSchema' structure. The 'formSchema' consists of a 'schema' object and an 'options' object, described under Properties.

You can choose from any question types listed at http://www.alpacajs.org/documentation.html. In that documentation, you will see that each field type - e.g., Checkbox, Radio, Text - has some 'Schema' properties and some 'Options' properties. The properties under 'Schema' will be defined in the 'schema' object of your formSchema. The properties under 'Options' will be defined in the 'options' object of your formSchema.

Many question types allow you to easily validate answers. For instance, for a "number" field you can set minimum and maximum values, and entries will automatically be required to be numeric (http://www.alpacajs.org/docs/fields/number.html). You can also either set `required:  true` in the schema->properties entry for this field OR set `validator:  required-field` in the options->fields entry if you want to require that the participant enters something. A validation error message will be displayed next to any fields that fail validation checks and the participant will not be able to proceed until these are addressed.

Alpacajs is fairly powerful, and you are essentially using it directly. In general, you can copy and paste any object passed to alpaca in the alpaca docs right in as your formSchema to see that example in action on Lookit. Not all features of alpaca are detailed here, but they can be used: e.g., advanced users can enter 'views' and 'data' in the formSchema to customize the layout of their forms and the initial data. A 'dataSource' may be specified under options to populate a question's potential answers (e.g., to load a list of countries from some other source rather than hard-coding it, or to provide checkboxes with vocabulary items from an externally-defined inventory).

You can also use alpacajs's "dependencies" and "conditional dependencies" functionality to set up fields that depend on other fields - e.g., asking if the child speaks any language besides English in the home and only if so displaying a dropdown to select the language(s), or asking if the child likes Elmo or Grover better and then asking a question specific to the preferred character. Or if you have questions only relevant to the birth mother of the child, you could ask if the participant is the birth mother and show those questions conditionally.

Formatting

Note that question titles are interpreted as HTML and can include images, audio/video elements, and inline CSS.

The form itself occupies a maximum of 800px horizontally and takes up 80% of the vertical height of the window (it will scroll to fit).

Current limitations

You are NOT able to provide custom functions (e.g. validators, custom dataSource functions) directly to the form-Schema.

What it looks like

Tell us about your pet!

★Age

★Name

a name...

What type of animal?

- ○ None
- ○ bird
- ○ cat
- ○ dog
- ○ fish
- ○ raccoon

Previous   Moving on...

More general functionality

Below is information specific to this particular frame. There may also be available parameters, events recorded, and data collected that come from the following more general sources:

- the *base frame* (things all frames do)

3.14.2 Examples

Simple pet survey

Here is an example of a reasonably simple survey using this frame:

```
"pet-survey": {
    "kind": "exp-lookit-survey",
    "formSchema": {
        "schema": {
            "type": "object",
            "title": "Tell us about your pet!",
            "properties": {
                "age": {
                    "type": "integer",
                    "title": "Age",
                    "maximum": 200,
                    "minimum": 0,
                    "required": true
                },
                "name": {
                    "type": "string",
                    "title": "Name",
                    "required": true
                },
                "species": {
                    "enum": [
                        "dog",
                        "cat",
                        "fish",
                        "bird",
                        "raccoon"
                    ],
                    "type": "string",
                    "title": "What type of animal?",
                    "default": ""
                }
            }
        },
        "options": {
            "fields": {
                "age": {
                    "numericEntry": true
                },
                "name": {
                    "placeholder": "a name..."
                },
                "species": {
                    "type": "radio",
```

(continues on next page)

```
                "message": "Seriously, what species??",
                "validator": "required-field"
            }
        }
    }
},
"nextButtonText": "Moving on..."
}
```

Reproducing the mood survey

And here is an example of re-implementing the exp-lookit-mood-questionnaire frame, using custom formatting, time-pickers, dependencies, and question groups.

```
"mood-survey": {
    "kind": "exp-lookit-survey",
    "formSchema": {
        "view": {
            "fields": {
                "/child/happy": {
                    "templates": {
                        "control": "<div>{{#if options.leftLabel}}<label class='label-
→left'>{{{options.leftLabel}}}</label>{{/if}}{{#control}}{{/control}}{{#if options.
→rightLabel}}<label class='label-right'>{{{options.rightLabel}}}</label>{{/if}}</div>
→"
                    }
                },
                "/child/active": {
                    "templates": {
                        "control": "<div>{{#if options.leftLabel}}<label class='label-
→left'>{{{options.leftLabel}}}</label>{{/if}}{{#control}}{{/control}}{{#if options.
→rightLabel}}<label class='label-right'>{{{options.rightLabel}}}</label>{{/if}}</div>
→"
                    }
                },
                "/child/rested": {
                    "templates": {
                        "control": "<div>{{#if options.leftLabel}}<label class='label-
→left'>{{{options.leftLabel}}}</label>{{/if}}{{#control}}{{/control}}{{#if options.
→rightLabel}}<label class='label-right'>{{{options.rightLabel}}}</label>{{/if}}</div>
→"
                    }
                },
                "/child/healthy": {
                    "templates": {
                        "control": "<div>{{#if options.leftLabel}}<label class='label-
→left'>{{{options.leftLabel}}}</label>{{/if}}{{#control}}{{/control}}{{#if options.
→rightLabel}}<label class='label-right'>{{{options.rightLabel}}}</label>{{/if}}</div>
→"
                    }
                },
                "/parent/energetic": {
                    "templates": {
                        "control": "<div>{{#if options.leftLabel}}<label class='label-
→left'>{{{options.leftLabel}}}</label>{{/if}}{{#control}}{{/control}}{{#if options.
→rightLabel}}<label class='label-right'>{{{options.rightLabel}}}</label>{{/if}}</div>
→"
```

```
                }
            },
            "/parent/parentHappy": {
                "templates": {
                    "control": "<div>{{#if options.leftLabel}}<label class='label-
→left'>{{{options.leftLabel}}}</label>{{/if}}{{#control}}{{/control}}{{#if options.
→rightLabel}}<label class='label-right'>{{{options.rightLabel}}}</label>{{/if}}</div>
→"
                }
            },
            "/parent/ontopofstuff": {
                "templates": {
                    "control": "<div>{{#if options.leftLabel}}<label class='label-
→left'>{{{options.leftLabel}}}</label>{{/if}}{{#control}}{{/control}}{{#if options.
→rightLabel}}<label class='label-right'>{{{options.rightLabel}}}</label>{{/if}}</div>
→"
                }
            }
        },
        "layout": {
            "bindings": {
                "child": "#child",
                "parent": "#parent",
                "lastEat": "#lastEat",
                "nextNap": "#nextNap",
                "napWakeUp": "#napWakeUp",
                "doingBefore": "#doingBefore",
                "usualNapSchedule": "#usualNapSchedule"
            },
            "template": "<div class='row exp-text exp-lookit-mood-questionnaire'>
→<h4>{{{options.formTitle}}}</h4><p>{{{options.introText}}}</p><div id='child'></div>
→<div id='parent'></div><div id='napWakeUp'></div><div id='usualNapSchedule'></div>
→<div id='nextNap'></div><div id='lastEat'></div><div id='doingBefore'></div></div>"
        },
        "parent": "bootstrap-edit"
    },
    "schema": {
        "type": "object",
        "properties": {
            "child": {
                "type": "object",
                "title": "How is your CHILD feeling right now?",
                "properties": {
                    "happy": {
                        "enum": [
                            "1",
                            "2",
                            "3",
                            "4",
                            "5",
                            "6",
                            "7"
                        ],
                        "required": true
                    },
                    "active": {
                        "enum": [
```

```
                                "1",
                                "2",
                                "3",
                                "4",
                                "5",
                                "6",
                                "7"
                        ],
                        "required": true
                    },
                    "rested": {
                        "enum": [
                                "1",
                                "2",
                                "3",
                                "4",
                                "5",
                                "6",
                                "7"
                        ],
                        "required": true
                    },
                    "healthy": {
                        "enum": [
                                "1",
                                "2",
                                "3",
                                "4",
                                "5",
                                "6",
                                "7"
                        ],
                        "required": true
                    }
                }
            },
            "parent": {
                "type": "object",
                "title": "How are YOU feeling right now?",
                "properties": {
                    "energetic": {
                        "enum": [
                                "1",
                                "2",
                                "3",
                                "4",
                                "5",
                                "6",
                                "7"
                        ],
                        "required": true
                    },
                    "parentHappy": {
                        "enum": [
                                "1",
                                "2",
                                "3",
```

```
                            "4",
                            "5",
                            "6",
                            "7"
                        ],
                        "required": true
                    },
                    "ontopofstuff": {
                        "enum": [
                            "1",
                            "2",
                            "3",
                            "4",
                            "5",
                            "6",
                            "7"
                        ],
                        "required": true
                    }
                }
            },
            "lastEat": {
                "title": "About how long ago did your child last eat or drink?",
                "required": true
            },
            "nextNap": {
                "title": "About how much longer until his/her next nap (or␣
→bedtime)?",
                "required": true
            },
            "napWakeUp": {
                "title": "About how long ago did your child last wake up from␣
→sleep or a nap?",
                "required": true
            },
            "doingBefore": {
                "title": "What was your child doing before this?",
                "required": true
            },
            "usualNapSchedule": {
                "enum": [
                    "yes",
                    "no",
                    "yes-overdue"
                ],
                "title": "Does your child have a usual nap schedule?",
                "required": true
            }
        },
        "dependencies": {
            "nextNap": [
                "usualNapSchedule"
            ]
        }
    },
    "options": {
        "fields": {
```

```
                "child": {
                    "fields": {
                        "happy": {
                            "type": "radio",
                            "order": 3,
                            "vertical": false,
                            "leftLabel": "Fussy",
                            "fieldClass": "aligned-radio-group",
                            "rightLabel": "Happy",
                            "optionLabels": [
                                "",
                                "",
                                "",
                                "",
                                "",
                                "",
                                ""
                            ]
                        },
                        "active": {
                            "type": "radio",
                            "order": 4,
                            "vertical": false,
                            "leftLabel": "Calm",
                            "fieldClass": "aligned-radio-group",
                            "rightLabel": "Active",
                            "optionLabels": [
                                "",
                                "",
                                "",
                                "",
                                "",
                                "",
                                ""
                            ]
                        },
                        "rested": {
                            "type": "radio",
                            "order": 1,
                            "vertical": false,
                            "leftLabel": "Tired",
                            "fieldClass": "aligned-radio-group",
                            "rightLabel": "Rested",
                            "optionLabels": [
                                "",
                                "",
                                "",
                                "",
                                "",
                                "",
                                ""
                            ]
                        },
                        "healthy": {
                            "type": "radio",
                            "order": 2,
                            "vertical": false,
```

```
                "leftLabel": "Sick",
                "fieldClass": "aligned-radio-group",
                "rightLabel": "Healthy",
                "optionLabels": [
                    "",
                    "",
                    "",
                    "",
                    "",
                    "",
                    ""
                ]
            }
        }
    },
    "parent": {
        "fields": {
            "energetic": {
                "type": "radio",
                "order": 1,
                "vertical": false,
                "leftLabel": "Tired",
                "fieldClass": "aligned-radio-group",
                "rightLabel": "Energetic",
                "optionLabels": [
                    "",
                    "",
                    "",
                    "",
                    "",
                    "",
                    ""
                ]
            },
            "parentHappy": {
                "type": "radio",
                "order": 3,
                "vertical": false,
                "leftLabel": "Upset",
                "fieldClass": "aligned-radio-group",
                "rightLabel": "Happy",
                "optionLabels": [
                    "",
                    "",
                    "",
                    "",
                    "",
                    "",
                    ""
                ]
            },
            "ontopofstuff": {
                "type": "radio",
                "order": 2,
                "vertical": false,
                "leftLabel": "Overwhelmed",
                "fieldClass": "aligned-radio-group",
```

```
                    "rightLabel": "On top of things",
                    "optionLabels": [
                        "",
                        "",
                        "",
                        "",
                        "",
                        "",
                        ""
                    ]
                }
            }
        },
        "lastEat": {
            "size": 10,
            "type": "time",
            "picker": {
                "useCurrent": "day"
            },
            "dateFormat": "HH:mm",
            "placeholder": "hours:minutes"
        },
        "nextNap": {
            "size": 10,
            "type": "time",
            "picker": {
                "useCurrent": "day"
            },
            "dateFormat": "HH:mm",
            "placeholder": "hours:minutes",
            "dependencies": {
                "usualNapSchedule": "yes"
            }
        },
        "napWakeUp": {
            "size": 10,
            "type": "time",
            "picker": {
                "useCurrent": "day"
            },
            "dateFormat": "HH:mm",
            "placeholder": "hours:minutes"
        },
        "doingBefore": {
            "type": "text",
            "placeholder": "examples: having lunch, playing outside, going to␣
↪the store with me"
        },
        "usualNapSchedule": {
            "sort": false,
            "type": "select",
            "hideNone": false,
            "noneLabel": "",
            "optionLabels": [
                "Yes",
                "No",
                "Yes, and he/she is already due for a nap"
```

```
                ],
                "removeDefaultNone": false
            }
        },
        "formTitle": "Mood Questionnaire",
        "introText": "How are you two doing? We really want to know: we're
→interested in how your child's mood affects which sorts of surprising physical
→events he/she notices. You can help us find out what babies are really learning as
→they get older... and what they already knew, but weren't calm and focused enough
→to show us!",
        "hideInitValidationError": true
    }
},
"nextButtonText": "Next"
}
```

3.14.3 Parameters

showPreviousButton [Boolean | `true`] Whether to show a 'previous' button

nextButtonText [String | `'Next'`] Text to display on the 'next frame' button

formSchema [Object] Object specifying the content of the form. This is in the same format as the example definition of the const 'schema' at http://toddjordan.github.io/ember-cli-dynamic-forms/#/demos/data: a schema and options are designated separately. Each field of the form must be defined in schema. Options may additionally be specified in options. This object has fields:

> **schema** [Object] The schema defines the fields in this form. It has the following properties:
>
> > **type** [String] This MUST be the string `'object'`.
> >
> > **title** [String] A form title for display
> >
> > **properties** [Object] An object defining the set of questions in this form and their associated data types, at minimum. Each key:value pair in this object is of the form `'FIELDNAME': {...}`. The `FIELDNAME` is something you select, like `age`; it should be unique within this form. The object contains at least 'type' and 'title' values, as well as any additional desired parameters that belong to the 'Schema' for the desired field described at http://www.alpacajs.org/documentation.html.
>
> **options** [Object] The options allow additional customization of the forms specified in the schema. This object should have a single key 'fields' mapping to an object. Each key:value pair in this object is of the form FIELDNAME:object, with FIELDNAMEs the same as in the schema. The potential parameters to use are those that belong to the 'Options' for the desired field described at http://www.alpacajs.org/documentation.html.

3.14.4 Data collected

The fields added specifically for this frame type are:

formSchema [Object] The same formSchema that was provided as a parameter to this frame, for ease of analysis if randomizing or iterating on experimental design.

formData [Object] Data corresponding to the fields defined in formSchema['schema']['properties']. The keys of formData are the FIELDNAMEs used there, and the values are the participant's responses. Note that if the participant does not answer a question, that key may be absent, rather than being present with a null value.

3.14.5 Events recorded

No events are recorded specifically by this frame.

3.15 exp-lookit-text

3.15.1 Overview

A frame to display simple text-only instructions, etc. to the participant.

What it looks like

Your baby, the physicist

Important: your child does not need to be with you until the videos begin. First, let's go over what will happen!

Some introductory text about this study.

Another paragraph about this study.

Next

More general functionality

Below is information specific to this particular frame. There may also be available parameters, events recorded, and data collected that come from the following more general sources:

- the *base frame* (things all frames do)

3.15.2 Examples

```
"study-intro": {
    "blocks": [
        {
            "emph": true,
            "text": "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
↪eiusmod tempor incididunt ut labore et dolore magna aliqua.",
            "title": "Lorem ipsum"
        },
        {
            "text": "Ut enim ad minim veniam, quis nostrud exercitation ullamco
↪laboris nisi ut aliquip ex ea commodo consequat."
            "listblocks": [
                {
                    "text": "Duis aute irure dolor in reprehenderit in voluptate
↪velit esse cillum dolore eu fugiat nulla pariatur."
                },
                {
                    "text": "Excepteur sint occaecat cupidatat non proident, sunt in
↪culpa qui officia deserunt mollit anim id est laborum."
                }
            ]
        },
        {
            "text": "Sed ut perspiciatis unde omnis iste natus error sit voluptatem
↪accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo
↪inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo."
        }
    ],
    "showPreviousButton": false,
    "kind": "exp-lookit-text"
}
```

3.15.3 Parameters

showPreviousButton [Boolean | `true`] Whether to show a 'previous' button

blocks [Array] Array of text blocks to display, rendered using *exp-text-block*.

3.15.4 Data collected

The fields added specifically for this frame type are:

<None>

3.15.5 Events recorded

The events recorded specifically by this frame are:

<None>

3.16 exp-lookit-video

3.16.1 Overview

Video display frame. This may be used for looking measures (looking time, preferential looking, etc.) as well as for brief filler between test trials or for displaying videos to older children or parents.

(Note: this frame replaced a now-deprecated exp-lookit-video frame, now called `exp-lookit-composite-video-trial`.)

This is very customizable: you can. . .

- position the video wherever you want on the screen, including specifying that it should fill the screen (while maintaining aspect ratio)

- choose the background color

- optionally specify audio that should play along with the video

- have the frame proceed automatically (`autoProceed`), or enable a Next button when the user can move on

- allow parents to press a key to pause the video (and then either restart when they un-pause, or move on to the next frame)

What it looks like
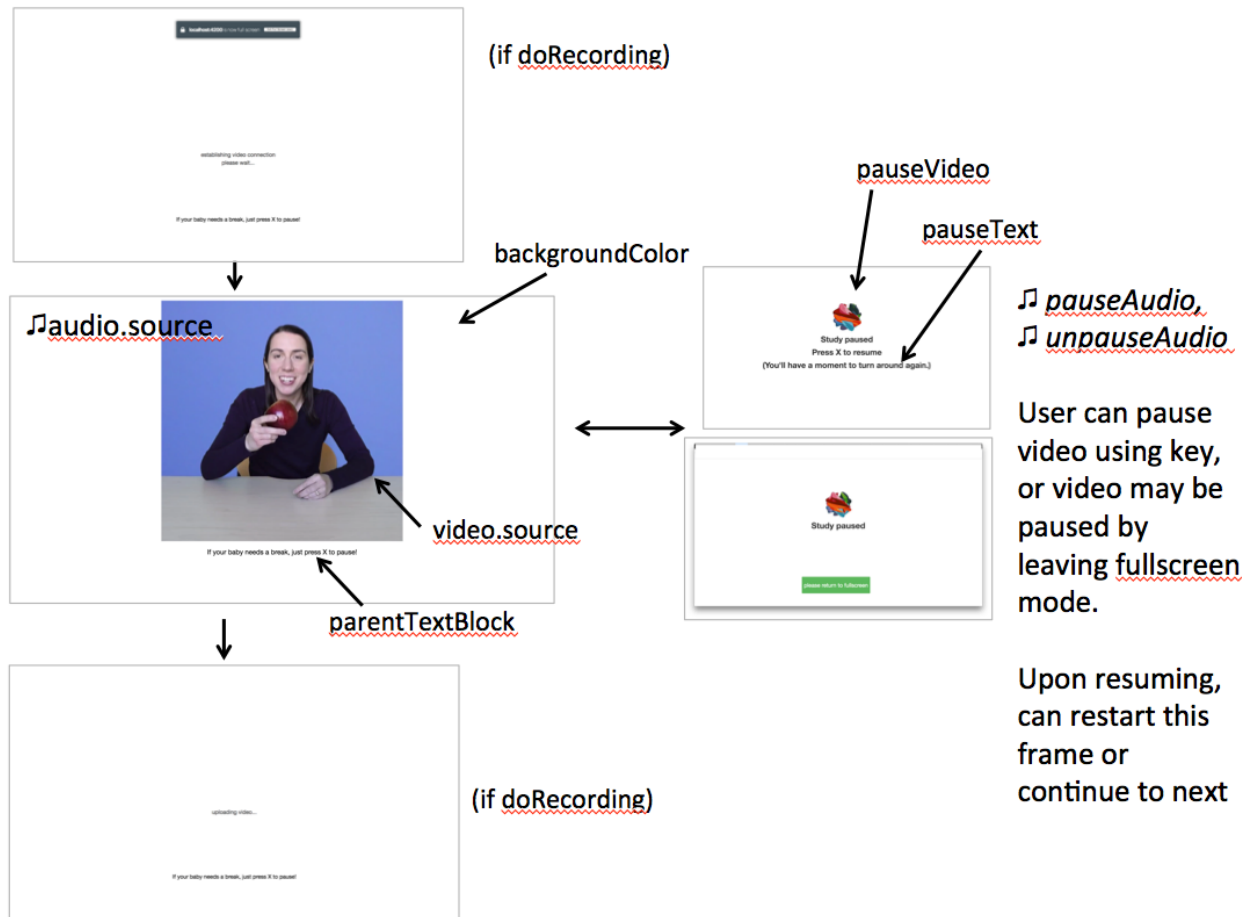
Specifying trial duration

There are several ways you can specify how long the trial should last. The frame will continue until ALL of the following are true:

- the video has been played all the way through `requireVideoCount` times
- the audio has been played all the way through `requireAudioCount` times
- `requiredDuration` seconds have elapsed since beginning the video

You do not need to use all of these. For instance, to play the video one time and then proceed, set `requireVideoCount` to 1 and the others to 0. You can also specify whether the audio and video should loop (beyond what is necessary to reach the required counts).

After the required number of video play-throughs, the video will either freeze or continue looping (depending on the `loop` property of your `video` parameter). Likewise, the audio will either stop or continue looping after the required number of audio play-throughs.

Recording

Video (and audio if provided) start as soon as any recording begins, or right away if there is no recording starting.

Pausing

If the user pauses using the `pauseKey`, or if the user leaves fullscreen mode, the study will be paused. If you don't want to allow pausing during the frame, set the `pauseKey` to an empty string (`""`).

While paused, the video/audio are stopped and not displayed, and instead a looping `pauseVideo` and text are displayed.

Leaving fullscreen mode will also pause the trial. For this reason, it's important to specify pause/unpause audio even if you plan to disable pausing!

Display

This frame is displayed fullscreen; if the frame before it is not, that frame needs to include a manual "next" button so that there's a user interaction event to trigger fullscreen mode. (Browsers don't allow us to switch to FS without a user event.)

Specifying where files are

Several of the parameters for this frame can be specified either by providing a list of full URLs and file types, or by providing just a filename that will be interpreted relative to the `baseDir`. These are: `audio` (source property), `pauseAudio`, `unpauseAudio`, `pauseVideo`, and `video` (source property). See the *expand-assets mixin* tool that this frame uses.

More general functionality

Below is information specific to this particular frame. There may also be available parameters, events recorded, and data collected that come from the following more general sources:

- the *base frame* (things all frames do)
- *video-record mixin*
- *expand-assets mixin*

3.16.2 Examples

This frame will play through a central video of Kim introducing an apple two times, then proceed.

```
"play-video-twice": {
    "kind": "exp-lookit-video",

    "video": {
        "top": 10,
        "left": 25,
        "loop": false,
        "width": 50,
        "source": "cropped_apple"
    },
```

(continues on next page)

```
        "backgroundColor": "white",
        "autoProceed": true,
        "parentTextBlock": {
            "text": "If your child needs a break, just press X to pause!"
        },

        "requiredDuration": 0,
        "requireAudioCount": 0,
        "requireVideoCount": 2,
        "doRecording": true,

        "pauseKey": "x",
        "pauseKeyDescription": "X",
        "restartAfterPause": true,
        "pauseAudio": "pause",
        "pauseVideo": "attentiongrabber",
        "pauseText": "(You'll have a moment to turn around again.)",
        "unpauseAudio": "return_after_pause",

        "baseDir": "https://www.mit.edu/~kimscott/placeholderstimuli/",
        "audioTypes": [
            "ogg",
            "mp3"
        ],
        "videoTypes": [
            "webm",
            "mp4"
        ]
}
```

This frame plays some audio announcing the next trial while an attention-grabber video loops. It doesn't record webcam video.

```
"announce-next-trial": {
    "kind": "exp-lookit-video",

    "audio": {
        "loop": false,
        "source": "video_01"
    },
    "video": {
        "top": 10,
        "left": 40,
        "loop": true,
        "width": 20,
        "source": "attentiongrabber"
    },
    "backgroundColor": "white",
    "autoProceed": true,
    "parentTextBlock": {
        "text": "If your child needs a break, just press X to pause!"
    },

    "requiredDuration": 0,
    "requireAudioCount": 1,
    "requireVideoCount": 0,
```

```
    "doRecording": false,

    "pauseKey": "x",
    "pauseKeyDescription": "X",
    "restartAfterPause": true,
    "pauseAudio": "pause",
    "pauseVideo": "attentiongrabber",
    "pauseText": "(You'll have a moment to turn around again.)",
    "unpauseAudio": "return_after_pause",

    "baseDir": "https://www.mit.edu/~kimscott/placeholderstimuli/",
    "audioTypes": [
        "ogg",
        "mp3"
    ],
    "videoTypes": [
        "webm",
        "mp4"
    ]
}
```

3.16.3 Parameters

video [Object] Object describing the video to show. It can have the following properties:

> **source** [String or Array] The location of the main video to play. This can be either an array of `{'src': 'https://...', 'type': '...'}` objects (e.g., to provide both webm and mp4 versions at specified URLS) or a single string relative to `baseDir/<EXT>/`.

> **left** [Number] left margin, as percentage of screen width. If none of left, width, top, and height is provided, the image is centered horizontally at its original size.

> **width** [Number] video width, as percentage of screen width. Note: in general only provide one of width and height; the other will be adjusted to preserve the video aspect ratio.

> **top** [Number] top margin, as percentage of video area height (i.e. 100 = whole screen, unless parent text or next button are shown). If no positioning parameters are provided, the image is centered vertically.

> **height** [Number] video height, as percentage of video area height. Note: in general only provide one of width and height; the other will be adjusted to preserve the video aspect ratio.

> **position** [String] set to 'fill' to fill the screen as much as possible while preserving aspect ratio. This overrides any left/width/top/height values.

> **loop** [Boolean] whether the video should loop, even after any `requireTestVideoCount` is satisfied.

audio [Object | {}] Object describing the audio to play along with video, if any. Can have properties:

> **source** [String or Object] Location of the audio file to play. This can either be an array of {src: 'url', type: 'MIMEtype'} objects, e.g. listing equivalent .mp3 and .ogg files, or can be a single string `filename` which will be expanded based on `baseDir` and `audioTypes` values (see `audioTypes`).

> **loop** [Boolean] whether the video audio loop, even after any `requireTestAudioCount` is satisfied.

autoProceed [Boolean|`true`] Whether to proceed automatically when video is complete / requiredDuration is achieved, vs. enabling a next button at that point.

> If true, the frame auto-advances after ALL of the following happen
>
> > (a) the requiredDuration (if any) is achieved, counting from the video starting
> >
> > (b) the video is played requireVideoCount times
> >
> > (c) the audio is played requireAudioCount times
>
> If false: a next button is displayed. It becomes possible to press 'next' only once the conditions above are met.

backgroundColor [String|`'white'`] Color of background. See https://developer.mozilla.org/en-US/docs/Web/CSS/color_value for acceptable syntax: can use color names ('blue', 'red', 'green', etc.), or rgb hex values (e.g. '#800080' - include the '#'). We recommend a light background if you need to see children's eyes.

pauseVideo [String or Array]

> **Video to show (looping) when trial is paused. As with the main video source, this can either be an array of** `{'src': 'https://...', 'type': '...'}` objects (e.g. providing both webm and mp4 versions at specified URLS) or a single string relative to `baseDir/<EXT>/`.

pauseAudio [String or Array] Audio to play [one time] upon pausing study, e.g. "Study paused." This can be either an array of `{src: 'url', type: 'MIMEtype'}` objects or a single string relative to `baseDir/<EXT>`.

unpauseAudio [String or Array] Audio to play [one time] when participant resumes the study, before actually resuming. E.g. this might give them a chance to get back in position. This can be either an array of `{src: 'url', type: 'MIMEtype'}` objects or a single string relative to `baseDir/<EXT>`.

pauseKey [String|`' '`] Key to pause the trial. Use an empty string, '', to not allow pausing using the keyboard. You can look up the names of keys at https://keycode.info. Default is the space bar (' ').

pauseKeyDescription [String|`'space'`] Parent-facing description of the key to pause the study. This is just used to display text "Press {pauseKeyDescription} to resume" when the study is paused.

restartAfterPause [Boolean|`true`] Whether to restart this frame upon unpausing, vs moving on to the next frame

pauseText [String|"(You'll have a moment to turn around again.)"] Text to show under "Study paused / Press space to resume" when study is paused.

requiredDuration [Number|`0`] Duration to require before proceeding, if any. Set if you want a time-based limit. E.g., setting requiredDuration to 20 means that the first 20 seconds of the video will be played, with shorter videos looping until they get to 20s. Leave out or set to 0 to play the video through to the end a set number of times instead.

requireVideoCount [Number|`1`] Number of times to play test video before moving on.

requireAudioCount [Number|`0`] Number of times to play test audio before moving on

doRecording [Boolean|`true`] Whether to do any (frame-specific) video recording during this frame. Set to false for e.g. last frame where just doing an announcement.

parentTextBlock [Object|`{}`] Text block to display to parent. Can have the following fields, each optional:

> **title** String title to display
>
> **text** String paragraph of text
>
> **css** Object object specifying any css properties to apply to this section, and their values - e.g. `{'color': 'gray', 'font-size': 'large'}`

3.16.4 Data collected

The fields added specifically for this frame type are:

videoShown [String] Source of video shown during this trial. Just stores first URL if multiple formats are offered.

audioPlayed [String] Source of audio played during this trial. Just stores first URL if multiple formats are offered.

hasBeenPaused [Boolean] Whether the video was paused at any point during the trial

3.16.5 Events recorded

The events recorded specifically by this frame are:

> **videoStarted** When video begins playing (recorded each time video starts if played through more than once)
>
> **videoStopped** When video completes playback (recorded each time if played more than once)
>
> **audioStarted** When audio begins playing (recorded each time video starts if played through more than once)
>
> **audioStopped** When audio completes playback (recorded each time if played more than once)
>
> **trialCompleted** When trial is complete and begins cleanup (may still then wait for video upload)
>
> **pauseTrial** When trial is paused
>
> **unpauseTrial** When trial is unpaused (actually proceeding to beginning or next frame, after unpauseAudio)
>
> **nextButtonEnabled** When all requirements for this frame are completed and next button is enabled (only recorded if autoProceed is false)

3.17 exp-lookit-video-assent

3.17.1 Overview

Video assent frame for Lookit studies for older children to agree to participation, separately from parental consent.

A series of assent form "pages" is displayed, each one displaying some combination of (a) an image or the participant's webcam view or a video, (b) audio, and (c) text. You can optionally record webcam video during the whole assent procedure or on the last page.

How the child assents

Once the family has viewed all pages, the child can answer a question about whether to participate. If they choose yes, they proceed; if they choose no, they are sent to the exit URL.

You can either simply have children click on "Yes" or "No," or you can add audio/video on the last page that instructs them to answer verbally, and do webcam recording on that page. For instance, you might show a video of yourself asking "Do you want to participate in this study? You can say "yes" or "no." Parents, once your child has answered, please click on their answer to the right."

Showing images, videos, webcam view, and audio

In general it is expected that only one of webcam view, video, and image will be provided per page, although it is ok to have only text or only text plus audio. If audio or video is provided for a page, the participant must finish it to proceed. (If both audio and video are provided they will be played simultaneously and both must finish for the participant to proceed.) They only need to complete the audio/video for a given page once, in case they navigate using the previous/next buttons.

Showing this frame only to older children

This frame can optionally be shown only when the child is at least N years old, in case some participants will need to give assent and others will rely only on parent consent.

What it looks like

Specifying where files are

Several of the parameters for this frame can be specified either by providing a list of full URLs and file types, or by providing just a filename that will be interpreted relative to the `baseDir`. See the *expand-assets mixin* tool that this frame uses.

More general functionality

Below is information specific to this particular frame. There may also be available parameters, events recorded, and data collected that come from the following more general sources:

- the *base frame* (things all frames do)
- *video-record mixin*
- *expand-assets mixin*

3.17.2 Example

```
"video-assent": {
    "kind": "exp-lookit-video-assent",
    "pages": [
        {
            "imgSrc": "two_cats.JPG",
            "altText": "two cats",
            "textBlocks": [
                {
                    "text": "My name is Jane Smith. I am a scientist who studies why
→children love cats."
                }
            ],
            "audio": "sample_1",
            "type": "audio/mp3"
        },
        {
            "imgSrc": "three_cats.JPG",
            "altText": "picture of sample game",
            "textBlocks": [
                {
                    "text": "In this study, you will play a game about cats."
                }
            ]
        },
        {
            "showWebcam": true,
            "textBlocks": [
                {
                    "text": "During the study, your webcam will record a video of you.
→ We will watch this video later to see how much you love cats."
                }
            ]
        }
    ],
    "baseDir": "https://www.mit.edu/~kimscott/placeholderstimuli/",
    "videoTypes": [
        "webm",
        "mp4"
    ],
    "participationQuestion": "Do you want to participate in this study?",
    "minimumYearsToAssent": 7
}
```

3.17.3 Parameters

pages [Array] A list of pages of assent form text/pictures/video for the participant to read through. Each has fields:

> **altText** [String] Alt-text used for the image displayed, if any
>
> **video** [String or Array] String indicating video path relative to baseDir, OR Array of {src: 'url', type: 'MIMEtype'} objects. Video will be displayed (with controls shown) and participant must complete to proceed.
>
> **audio** [String or Array] String indicating audio path relative to baseDir, OR Array of {src: 'url', type: 'MIMEtype'} objects. Audio will be played (with controls shown) and participant must complete to proceed.
>
> **imgSrc** [String] URL of image to display; can be full path or relative to baseDir
>
> **textBlocks** [Array] list of text blocks to show on this page, processed by *exp-text-block*. Can use HTML.
>
> **showWebcam** [Boolean] Whether to display the participant webcam on this page

nextStimulusText [String | `'Next'`] Text on the button to proceed to the next example video/image

previousStimulusText [String | `'Previous'`] Text on the button to proceed to the previous example video/image

recordLastPage [Boolean | `false`] Whether to record webcam video on the last page

recordWholeProcedure [Boolean | `false`] Whether to record webcam video during the entire assent frame (if true, overrides recordLastPage)

participationQuestion [String | `'Do you want to participate in this study?'`] Text of the question to ask about whether to participate. Answer options are Yes/No; No means study will stop, Yes means it will proceed.

minimumYearsToAssent [Number | `0`] How many years old the child has to be for this page to be shown. If child is younger, the page is skipped. Leave at 0 to always show. This is an age in 'calendar years' - it will line up with the child's birthday, regardless of leap years etc.

3.17.4 Data collected

The fields added specifically for this frame type are:

assentFormText [String] the exact text shown in the assent document during this frame

childResponse [String] The child's response to the assent question - Yes or No

3.17.5 Events recorded

The events recorded specifically by this frame are:

> **nextAssentPage** Participant proceeded to next assent page
>
> > **pageNumber** [Number] which assent page was viewed (zero-indexed)
>
> **previousAssentPage** Participant returned to previous assent page
>
> > **pageNumber** [Number] which assent page was viewed (zero-indexed)
>
> **assentQuestionSubmit** Participant submitted assent question answer
>
> > **childResponse** [String] child response submitted ('Yes' or 'No')
>
> **downloadAssentForm** When participant downloads assent form

---

3.18 exp-lookit-video-consent

3.18.1 Overview

Video consent frame for Lookit studies, with consent document displayed at left and instructions to start recording, read a statement out loud, and send. A standard consent document is displayed, with additional study-specific information provided by the researcher, in accordance with the Lookit terms of use.

The consent document can be downloaded as a PDF document by the participant.

Researchers can select from the following named templates:

- *consent_001*: Original Lookit consent document (2019)

- *consent_002*: Added optional GDPR section and research subject rights statement

- *consent_003*: Same as consent_002 except that the 'Payment' section is renamed 'Benefits, risks, and payment' for institutions that prefer that

- *consent_004*: Same as consent_003 except that sentences in 'Data collection and webcam recording' are rearranged to make it clearer what happens if you withdraw video data, and the prompt says to read "out loud (or in ASL)" instead of just "out loud" for accessibility.

Looking up exact templates

To look up the exact text of each consent template for your IRB protocol, and to understand the context for each piece of text to be inserted, please see the templates.

What it looks like

More general functionality

Below is information specific to this particular frame. There may also be available parameters, events recorded, and data collected that come from the following more general sources:

- the *base frame* (things all frames do)
- *video-record mixin*
- *expand-assets mixin*

3.18.2 Example

```
"video-consent": {
    "kind": "exp-lookit-video-consent",
    "template": "consent_003",
    "PIName": "Jane Smith",
    "institution": "Science University",
    "PIContact": "Jane Smith at 123 456 7890",
    "purpose": "Why do babies love cats? This study will help us find out whether
→babies love cats because of their soft fur or their twitchy tails.",
```

<div align="right">(continues on next page)</div>

```
    "procedures": "Your child will be shown pictures of lots of different cats, along
→with noises that cats make like meowing and purring. We are interested in which
→pictures and sounds make your child smile. We will ask you (the parent) to turn
→around to avoid influencing your child's responses.",
    "payment": "After you finish the study, we will email you a $5 BabyStore gift
→card within approximately three days. To be eligible for the gift card your child
→must be in the age range for this study, you need to submit a valid consent
→statement, and we need to see that there is a child with you. But we will send a
→gift card even if you do not finish the whole study or we are not able to use your
→child's data! There are no other direct benefits to you or your child from
→participating, but we hope you will enjoy the experience. There are no anticipated
→risks associated with participating.",
    "datause": "We are primarily interested in your child's emotional reactions to
→the images and sounds. A research assistant will watch your video to measure the
→precise amount of delight in your child's face as he or she sees each cat picture.",
    "gdpr": false,
    "research_rights_statement": "You are not waiving any legal claims, rights or
→remedies because of your participation in this research study.  If you feel you
→have been treated unfairly, or you have questions regarding your rights as a
→research subject, you may contact the [IRB NAME], [INSTITUTION], [ADDRESS/CONTACT]",
    "additional_segments": [
        {
            "title": "US Patriot Act Disclosure",
            "text": "[EXAMPLE ONLY, PLEASE REMOVE ADDITIONAL_SEGMENTS UNLESS YOU
→NEED THEM.] Lookit is a U.S. organization and all information gathered from the
→website is stored on servers based in the U.S. Therefore, your video recordings are
→subject to U.S. laws, such as the US Patriot Act. This act allows authorities
→access to the records of internet service providers. If you choose to participate
→in this study, you understand that your video recording will be stored and accessed
→in the USA. The security and privacy policy for Lookit can be found at the
→following link: <a href='https://lookit.mit.edu/privacy/' target='_blank'>https://
→lookit.mit.edu/privacy/</a>."
        }
    ]
    }
}
```

3.18.3 Parameters

template [String | `'consent_001'`] Which consent document template to use. If you are setting up a new study, we recommend using the most recent (highest number) of these options. Options: `consent_001`, `consent_002`, `consent_003`, `consent_004`.

PIName Name of PI running this study

institution Name of institution running this study (if ambiguous, list institution whose IRB approved the study)'

PIContact Contact information for PI or lab in case of participant questions or concerns. This will directly follow the phrase "please contact", so format accordingly: e.g., "the XYZ lab at xyz@science.edu" or "Mary Smith at 123 456 7890".

purpose Brief description of purpose of study - 1-2 sentences that describe what you are trying to find out. Language should be as straightforward and accessible as possible! E.g., "Why do babies love cats? This study will help us find out whether babies love cats because of their soft fur or their twitchy tails."

procedures Brief description of study procedures. For consent templates 001 and 002, this should include any risks or a statement that there are no anticipated risks. (For consent template 003, that is included in *payment*). We add

a statement about the duration (from your study definition) to the start (e.g., "This study takes about 10 minutes to complete"), so you don't need to include that. It can be in third person or addressed to the parent. E.g., "Your child will be shown pictures of lots of different cats, along with noises that cats make like meowing and purring. We are interested in which pictures and sounds make your child smile. We will ask you (the parent) to turn around to avoid influencing your child's responses. There are no anticipated risks associated with participating."

prompt_all_adults [Boolean | `false`] Whether to include an addition step #4 prompting any other adults present to read a statement of consent (I have read and understand the consent document. I also agree to participate in this study.)

Please only set this to true if your IRB requires it.

payment Statement about payment/compensation for participation, including a statement that there are no additional benefits anticipated to the participant. E.g., "After you finish the study, we will email you a $5 BabyStore gift card within approximately three days. To be eligible for the gift card your child must be in the age range for this study, you need to submit a valid consent statement, and we need to see that there is a child with you. But we will send a gift card even if you do not finish the whole study or we are not able to use your child's data! There are no other direct benefits to you or your child from participating, but we hope you will enjoy the experience."

For consent templates 003+, this section is titled Benefits, risks, and payment; it should include information about risks as well.

datause Study-specific data use statement (optional). This will follow the following more general text: "The research group led by [PIName] at [institution] will have access to video and other data collected during this session. We will also have access to your account profile, demographic survey, and the child profile for the child who is participating, including changes you make in the future to any of this information. We may study your child's responses in connection with his or her previous responses to this or other studies run by our group, siblings' responses to this or other studies run by our group, or demographic survey responses." You may want to note what measures you will actually be coding for (looking time, facial expressions, parent-child interaction, etc.) and other more specific information about your use of data from this study here. For instance, you would note if you were building a corpus of naturalistic data that may be used to answer a variety of questions (rather than just collecting data for a single planned study).

gdpr [Boolean | `false`] Whether to include a section on GDPR; only used in template consent_002 + .

gdpr_personal_data [String] List of types of personal information collected, for GDPR section only. Do not include special category information, which is listed separately.

gdpr_sensitive_data [String] List of types of special category information collected, for GDPR section only. Include all that apply: racial or ethnic origin; political opinions; religious or philosophical beliefs; trade union membership; processing of genetic data; biometric data; health data; and/or sex life or sexual orientation information

research_rights_statement [String] Statement about rights of research subjects and how to contact IRB. Used only in template consent_002+. For instance, MIT's standard language is: You are not waiving any legal claims, rights or remedies because of your participation in this research study. If you feel you have been treated unfairly, or you have questions regarding your rights as a research subject, you may contact [CONTACT INFO].

additional_segments [Array] List of additional custom sections of the consent form, e.g. US Patriot Act Disclosure. These are subject to Lookit approval and in general can only add information that was true anyway but that your IRB needs included; please contact us before submitting your study to check.

Each section can have fields:

> **title** [String] title of section
>
> **text** [String] text of section

3.18.4 Data collected

The fields added specifically for this frame type are:

consentFormText the exact text shown in the consent document during this frame

3.18.5 Events recorded

The events recorded specifically by this frame are:

> **downloadConsentForm** When participant downloads consent form

3.19 exp-lookit-video-infant-control

3.19.1 Overview

Infant-controlled version of the *exp-lookit-video* frame. This works the same way as `exp-lookit-video` except that you can enable the parent to:

- end the trial by pressing the `endTrialKey` key

- hold down the `lookawayKey` (or the mouse button) to indicate that the child is not looking; the trial will automatically end after the lookaway criterion is met. If the 'lookawayTone' is not 'none' a noise is played while the child is looking away to help the parent know the looking coding is working.

You can disable either of these behaviors by setting the corresponding key to `''`.

The frame will still end when it would have anyway if neither of these things happen! For instance, if you would have looped the video for 30 seconds, then after 30 seconds the frame will move on, serving as a "ceiling" on looking time.

Lookaway criterion

You have two options for how to determine when the child has looked away long enough to proceed.

1. Set the `lookawayType` to `"total"` to accumulate lookaway time until the child has looked away for a total of *lookawayThreshold* seconds. (For instance, if the `lookawayThreshold` is 2, then the trial will end after the child looks away for 0.5s, then 1s, then 0.5s.) 2. Set the `lookawayType` to `"continuous"` to require that the child look away for a continuous `lookawayThreshold`-second interval. (For instance, if the `lookawayThreshold` is 2, then the child might look away for 1s, 1.5s, and 1s but the trial would continue until she looked away for 2s.)

When looking time is measured

The looking time measurement begins only when the video starts, not while a video connection is established.

If a *lookawayKey* is defined, lookaways are recorded the entire time the frame is running. However, the looking time measurement only starts once video begins playing (e.g., not during webcam connection). Lookaways at the very start of the video don't count! If the child is not looking as the video begins, the measurement begins once they look for the first time.

If the trial is paused, parent control of the trial is also paused; the looking time measurement begins fresh when restarting.

3.19.2 Examples

This frame will play through a central video of Kim introducing an apple up to five times. Once the child looks away for more than 2 s total, as coded by the parent holding down the P key, it will proceed.

```
"play-video-five-times": {
    "kind": "exp-lookit-video-infant-control",
    "lookawayKey": "p",
    "lookawayType": "total",
    "lookawayThreshold": 2,
    "endTrialKey": "q",
    "lookawayTone": "noise",
    "lookawayToneVolume": 0.25,

    "audio": {
        "loop": false,
        "source": "peekaboo"
    },
    "video": {
        "top": 10,
        "left": 25,
        "loop": true,
        "width": 50,
        "source": "cropped_apple"
    },
    "backgroundColor": "white",
    "autoProceed": true,
    "parentTextBlock": {
        "text": "If your child needs a break, just press X to pause!"
    },
    "requiredDuration": 0,
    "requireAudioCount": 0,
    "requireVideoCount": 5,
    "restartAfterPause": true,
    "pauseKey": "x",
    "pauseKeyDescription": "X",
    "pauseAudio": "pause",
    "pauseVideo": "attentiongrabber",
    "pauseText": "(You'll have a moment to turn around again.)",
    "unpauseAudio": "return_after_pause",
    "doRecording": true,
    "baseDir": "https://www.mit.edu/~kimscott/placeholderstimuli/",
    "audioTypes": [
        "ogg",
        "mp3"
    ],
    "videoTypes": [
        "webm",
        "mp4"
    ]
}
```

3.19.3 Parameters

The parameters for this frame are the same as for *exp-lookit-video*, plus the additional parameters provided by the *infant-controlled-timing mixin*.

3.19.4 Data collected

This frame collects the same data as *exp-lookit-video*, plus the additional data provided by the *infant-controlled-timing mixin*.

3.19.5 Events recorded

This frame records the same events as *exp-lookit-video*, plus the additional events recorded by the *infant-controlled-timing mixin*.

3.20 exp-lookit-webcam-display

3.20.1 Overview

A frame to display the user's webcam stream, along with a small amount of optional text. Expected use is as a break during an experiment, e.g. to check positioning, but could also be used as a lightweight frame for data collection or during setup.

Not fullscreen by default, but can be displayed fullscreen as shown in example below. Can optionally record video.

What it looks like

More general functionality

Below is information specific to this particular frame. There may also be available parameters, events recorded, and data collected that come from the following more general sources:

- the *base frame* (things all frames do)

- *video-record mixin*

3.20.2 Example

```
"webcam-display-break": {
    "kind": "exp-lookit-webcam-display",
    "blocks": [
        {
            "title": "Let's take a quick break",
            "listblocks": [
                {
                    "text": "Please check that your child is still visible"
                },
                {
                    "text": "You can make some silly faces!"
                }
            ]
        }
    ],
    "nextButtonText": "Next",
    "showPreviousButton": false,
    "displayFullscreenOverride": true,
    "startRecordingAutomatically": false
}
```

3.20.3 Parameters

blocks [Array] Array of blocks specifying text/images of instructions to display, rendered by *exp-text-block*.

startRecordingAutomatically [Boolean | `false`] Whether to automatically begin recording upon frame load

nextButtonText Text to display on the 'next frame' button

showPreviousButton Whether to show a 'previous' button

3.20.4 Data collected

The fields added specifically for this frame type are:

<None>

3.20.5 Events recorded
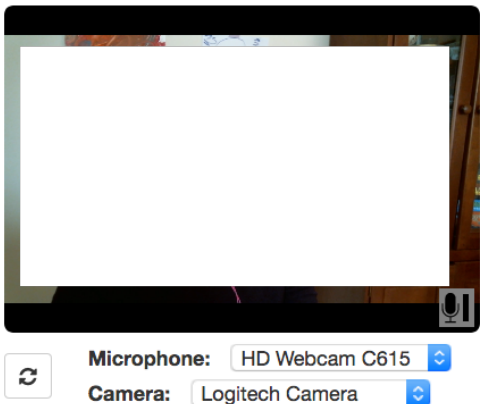
The events recorded specifically by this frame are:

<None>

3.21 exp-video-config

3.21.1 Overview

Video configuration frame guiding user through making sure permissions are set appropriately and microphone is working, with troubleshooting text. Almost all content is hard-coded, to provide a general-purpose technical setup frame.

What it looks like

More general functionality

Below is information specific to this particular frame. There may also be available parameters, events recorded, and data collected that come from the following more general sources:

- the *base frame* (things all frames do)

- *video-record mixin*

3.21.2 Examples

This frame just shows the standard video config frame.

```
"video-config": {
    "kind": "exp-video-config",
    "troubleshootingIntro": ""
}
```

This frame shows a friendly message offering tech support by the lab via phone.

```
"video-config": {
    "kind": "exp-video-config",
    "troubleshootingIntro": "If you're having any trouble getting your webcam set up,
      please feel free to call the XYZ lab at (123) 456-7890 and we'd be glad to
      help you out!"
}
```

3.21.3 Parameters

troubleshootingIntro [String | ''] Text to show as the introduction to the troubleshooting tips section

3.21.4 Data collected

The fields added specifically for this frame type are:

<None>

3.21.5 Events recorded

The events recorded specifically by this frame are:

<None>>

3.22 exp-video-config-quality

3.22.1 Overview

Video configuration frame showing webcam view at right and instructions for checking video quality for preferential looking setup at left. Some default content is hard-coded to provide a reasonable set of instructions for preferential looking setups.

Optionally, participants can be required to check off each item before they can proceed to the next frame. If *require-ItemConfirmation* is true (default), then the 'next' button will appear disabled until the participant has checked off all buttons, although if they click it anyway they will get an informative warning and the instructions section will scroll to the first unchecked item.

Participants can also be optionally required to create and view a short recording, e.g. to check their child will be audible or their child's eyes will be visible in a particular position.

What it looks like

More general functionality

Below is information specific to this particular frame. There may also be available parameters, events recorded, and data collected that come from the following more general sources:

- the *base frame* (things all frames do)
- *video-record mixin*

3.22.2 Examples

This frame will show the default hard-coded instructions for setup:

```
"video-quality": {
    "kind": "exp-video-config-quality"
}
```

Here the researcher provides specific items to allow customizing the instructions/pictures:

```
"video-quality": {
    "kind": "exp-video-config-quality",
    "title": "Webcam setup for preferential looking",
    "introText": "We'll be analyzing where your child chooses to look during the␣
→videos--but only if we can tell where that is! Please check each of the following␣
→to ensure we're able to use your video:",
    "requireItemConfirmation": true,
    "completedItemText": "Did it!",
    "instructionBlocks": [
            {
                "text": "<strong>Make sure the webcam you're using is roughly␣
→centered</strong> relative to this monitor. This makes it much easier for us to␣
→tell whether your child is looking to the left or right!",
                "image": {
                    "src": "https://s3.amazonaws.com/lookitcontents/website/
→centering.png",
                    "alt": "Example images of using centered external webcam on␣
→monitor or built-in webcam on laptop."
                }
            },
            {
                "text": "<strong>Turn off any other monitors</strong> connected␣
→to your computer, besides the one with the centered webcam. (If there's just one␣
→monitor, you're all set!)",
                "image": {
                    "src": "https://s3.amazonaws.com/lookitcontents/website/
→monitors.png",
                    "alt": "Example images showing laptop screen turned off if␣
→using external monitor and webcam, or external monitor turned off if using built-in␣
→webcam and laptop screen."
                }
            },
            {
                "text": "Check the lighting by making sure you can <strong>
→clearly see your own eyes</strong> on the webcam view to the right. You may need to␣
→either turn on a light or reduce light coming from behind you.",
                "image": {
                    "src": "https://s3.amazonaws.com/lookitcontents/website/
→lighting.png",
                    "alt": "Example images showing good lighting, room too dark,␣
→and backlit scene where eyes are not visible."
                }
            },
            {
                "text": "If it's practical, <strong>minimize exciting things</
→strong> that are visible behind or to the side of the screen--for instance, by␣
→facing a wall instead of the kitchen. (If this isn't practical for you, don't worry␣
→about it--just check the box!)",
```
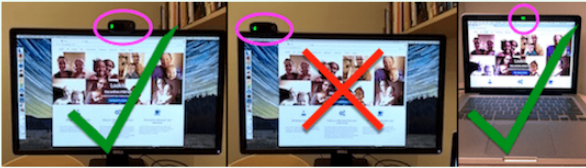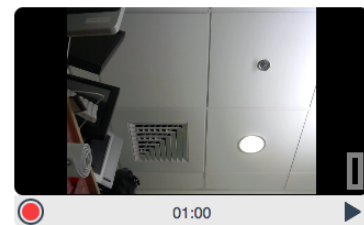
<div align="right">(continues on next page)</div>

```
                    "image": {
                        "src": "https://s3.amazonaws.com/lookitcontents/website/
↪distractions.png",
                        "alt": "Example images showing a child and puppy next to the␣
↪computer, versus a computer just on its own."
                    }
                },
                {
                    "text": "During the study, we'll ask that you sit facing away␣
↪from the monitor, holding your child over your shoulder, like this. (More on that␣
↪in a moment!) <strong>Make sure the webcam is angled up or down enough that your␣
↪child's eyes are visible in this position</strong>. If you're not sure if your child
↪'s eyes will be visible, you can make a short recording to check!",
                    "image": {
                        "src": "https://s3.amazonaws.com/lookitcontents/website/over_
↪shoulder.jpg",
                        "alt": "Example image showing a dad holding his child looking␣
↪over his shoulder."
                    }
                }
            ],
        "requireTestVideo": true,
        "showRecordMenu": true,
        "recordingInstructionText": "You should be able to see your camera view above.
↪ You can create and view a short recording to see how your setup looks."
}
```

3.22.3 Parameters

introText [String] Text to show as the introduction to the list of things to check. Can include HTML.

title [String] Title to display to participant

requireItemConfirmation [Boolean | `true`] Whether to show checkboxes under each instruction item and require participant to check them off to proceed.

requireTestVideo [Boolean | `true`] Whether to require participant to make and view a test video. Ignored if showRecordMenu is false.

showRecordMenu [Boolean | `true`] Whether to display record/replay menu to participant. If false, requireTestVideo value is ignored.

recordingInstructionText [String] Text to show below the webcam view. For instance, you might instruct families to make a short recording in the position they will be in for the experiment, and make sure that the infant's eyes are visible or that the child is audible. HTML is allowed.

completedItemText [String | `'Did it!'`] Text to show next to instructions checkboxes, if participant is required to check off each instruction (see requireItemConfirmation). Ignored if requireItemConfirmation is false.

instructionBlocks [Array] List of instruction segments to display to participant. Rendered using *exp-text-block*.

3.22.4 Data collected

The fields added specifically for this frame type are:

<None>

3.22.5 Events recorded

The events recorded specifically by this frame are:

<None>

MIXINS

Some frames use "mixins" that provide some shared functionality. For instance, a few frames use the "infant-controlled-timing" mixin that allows you to specify a button for the parent to press to indicate the child isn't looking, and move on after a specified lookaway time. Each frame will say what mixins it uses, but to keep things concise, you'll find information about what parameters to include and what data will be collected here.

4.1 expand-assets mixin

4.1.1 Overview

Frames that use this mixin allow you to specify some media locations (i.e., where your audio, video, or image is located) relative to a base directory with a specific file structure, rather than always giving the full URL. This can make your protocol configuration easier to read, reason about, and share productively (you can provide a copy of your stimuli directory to someone looking to replicate your work, for instance).

Directory structure

The files inside your base directory must be structured in a particular way for the experiment runner to find them!

Inside your base directory, you should have an `img` directory for images, and then one directory for each audio or video format you use - named exactly the same as the filename extension, like `mp3` or `mp4`.

Here's an example of what your directory might look like if you have mp3 and mp4 files:

```
baseDir
    img
        cat.jpg
        dog.jpg
    mp3
        meow.mp3
        ruff.mp3
    mp4
        cat_yawning.mp4
        dog_playing.mp4
```

If you choose to provide multiple file formats for your media files, the alternate versions should have exactly the same filenames except for the extensions. Here's an example like the above, except that we have both mp3 and ogg files for audio, and both mp4 and webm for video:

```
baseDir
    img
        cat.jpg
        dog.jpg
    mp3
        meow.mp3
        ruff.mp3
    ogg
        meow.ogg
        ruff.ogg
    mp4
        cat_yawning.mp4
        dog_playing.mp4
    webm
        cat_yawning.webm
        dog_playing.webm
```

Images

Include extensions in your image filenames when specifying them on a frame that uses the expand-assets mixin. Anything without `://` in the string will be assumed to be a relative image source, and the experiment runner will look in your base directory's `img` directory. For instance, the following two image specifications would be equivalent for specifying which image to use during calibration in the *exp-lookit-calibration* frame:

```
"calibrationImage": "cat.jpg"
"baseDir": "MYBASEDIR"
```

```
"calibrationImage": "MYBASEDIR/img/cat.jpg"
```

"MYBASEDIR" is a placeholder for whatever the URL to your base directory is - it might be something like `https://raw.githubusercontent.com/kimberscott/lookit-stimuli-template/master/` if you're using the GitHub stimuli template, or something like `https://www.mit.edu/~kimscott/placeholderstimuli/` if you're using your university's web hosting.

Audio files

Do not include the extensions when specifying audio files relative to the base directory. Specify all the audio types that are available (all the subdirectories you have for audio extensions) in the `audioTypes` parameter. If you wanted had a directory structure like the first one above, with mp3 files only for audio, you could specify the "meow.mp3" file to use during calibration in the *exp-lookit-calibration* frame:

```
"calibrationAudio": "meow",
"audioTypes": [ "mp3" ],
"baseDir": "MYBASEDIR"
```

If you have both mp3 and ogg file formats, you would instead write:

```
"calibrationAudio": "meow",
"audioTypes": [ "mp3", "ogg" ],
"baseDir": "MYBASEDIR"
```

Alternately, you can still provide full paths to your audio files. To provide full paths to audio files, you enter a **list** of "sources". Each element in the list provides both the URL and the file type, like this:

```
[
    {
        "src": "https://stimuli.org/myAudioFile.mp3",
        "type": "audio/mp3"
    },
    {
        "src": "https://stimuli.org/myAudioFile.ogg",
        "type": "audio/ogg"
    }
]
```

If you only have one file type, you would still need to provide a list, but it would only have one thing in it:

```
[
    {
        "src": "https://stimuli.org/myAudioFile.mp3",
        "type": "audio/mp3"
    }
]
```

Video files

Do not include the extensions when specifying video files relative to the base directory. Specify all the video types that are available (all the subdirectories you have for video extensions) in the `videoTypes` parameter. If you wanted had a directory structure like the first one above, with mp4 files only for mp4, you could specify the "cat_yawning.mp4" file to use during calibration in the *exp-lookit-calibration* frame:

```
"calibrationVideo": "cat_yawning",
"videoTypes": [ "mp4" ],
"baseDir": "MYBASEDIR"
```

If you have both mp4 and webm file formats, you would instead write:

```
"calibrationVideo": "cat_yawning",
"videoTypes": [ "mp4", "webm" ],
"baseDir": "MYBASEDIR"
```

Alternately, you can still provide full paths to your video files. To provide full paths to video files, you enter a **list** of "sources". Each element in the list provides both the URL and the file type, like this:

```
[
    {
        "src": "https://stimuli.org/myVideoFile.mp4",
        "type": "video/mp4"
    },
    {
        "src": "https://stimuli.org/myVideoFile.webm",
        "type": "video/webm"
    }
]
```

If you only have one file type, you would still need to provide a list, but it would only have one thing in it:

```
[
    {
```

```
        "src": "https://stimuli.org/myVideoFile.mp4",
        "type": "video/mp4"
    }
]
```

The Lookit stimuli template on Github

One option for hosting your stimuli is to put them in a GitHub repo which has the directory structure described above already built-in. You can make your own copy by following the directions at Lookit stimuli template repo. This is best for smallish stimuli (not for long video files >10MB). For a discussion of other options for hosting your files, see the main Lookit docs.

Troubleshooting

If media files aren't displaying as expected, the most important thing to do is to check the browser console for relevant error messages. Generally you will see a 404 error saying that a file wasn't found, so you can check where the experiment runner is *looking* for the file to understand if there's a discrepancy between that and where you have it.

4.1.2 Parameters

If your frame uses the expand-assets mixin, you can specify the following in addition to frame-specific parameters:

baseDir [String] Base directory for where to find stimuli. Any image src values that are not full paths will be expanded by prefixing with `baseDir` + `img/`. Any audio/video src values provided as strings rather than objects with `src` and `type` will be expanded out to `baseDir/avtype/[stub].avtype`, where the potential avtypes are given by `audioTypes` and `videoTypes`.

> `baseDir` should include a trailing slash (e.g., *http://stimuli.org/myexperiment/*); if a value is provided that does not end in a slash, one will be added.

audioTypes [Array | `['mp3', 'ogg']`] List of audio types to expect for any audio specified just with a string rather than with a list of src/type objects. If audioTypes is *['typeA', 'typeB']* and an audio source is given as *intro*, the audio source will be expanded out to

```
[
    {
        src: 'baseDir' + 'typeA/intro.typeA',
        type: 'audio/typeA'
    },
    {
        src: 'baseDir' + 'typeB/intro.typeB',
        type: 'audio/typeB'
    }
]
```

videoTypes [Array | `['mp4', 'webm']`] List of video types to expect for any audio specified just with a string rather than with a list of src/type objects. If videoTypes is *['typeA', 'typeB']* and a video source is given as *intro*, the video source will be expanded out to

```
[
    {
        src: 'baseDir' + 'typeA/intro.typeA',
```

```
        type: 'video/typeA'
    },
    {

        src: 'baseDir' + 'typeB/intro.typeB',
        type: 'video/typeB'
    }
]
```

4.1.3 Data collected

No additional data is collected.

4.1.4 Events recorded

No additional events are recorded.

4.2 infant-controlled-timing mixin

4.2.1 Overview

This mixin provides shared functionality for frames that allow the parent to live-code infant looking to determine when to proceed. Frames using this mixin allow the parent to:

- end the trial by pressing the `endTrialKey` key

- hold down the `lookawayKey` (or the mouse button) to indicate that the child is not looking; the trial will automatically end after the lookaway criterion is met. If the 'lookawayTone' is not 'none' a noise is played while the child is looking away to help the parent know the looking coding is working.

You can disable either of these behaviors by setting the key to `''`.

The frame will still end when it would have anyway if neither of these things happen! For instance, if you would have displayed an image for 30 seconds, then after 30 seconds the frame will move on, serving as a "ceiling" on looking time.

Lookaway criterion

You have two options for how to determine when the child has looked away long enough to proceed.

1. Set the `lookawayType` to `"total"` to accumulate lookaway time until the child has looked away for a total of *lookawayThreshold* seconds. (For instance, if the `lookawayThreshold` is 2, then the trial will end after the child looks away for 0.5s, then 1s, then 0.5s.) 2. Set the `lookawayType` to `"continuous"` to require that the child look away for a continuous `lookawayThreshold`-second interval. (For instance, if the `lookawayThreshold` is 2, then the child might look away for 1s, 1.5s, and 1s but the trial would continue until she looked away for 2s.)

4.2.2 Parameters

Any frame that uses this mixin will accept the following parameters in addition to the regular frame parameters:

lookawayType [String | `'total'`] Type of lookaway criterion. Must be either 'total' (to count total lookaway time) or 'continuous' (to count only continuous lookaway time). Whichever criterion type is used, only lookaways after the first look to the screen are considered.

lookawayThreshold [Number | `2`] Lookaway threshold in seconds. How long does the child need to look away before the trial ends? Depending on the lookawayType, this will refer either to the total amount of time the child has looked away since their first look to the screen, or to the length of a single continuous lookaway.

lookawayKey [String | `'p'`] Key parent should press to indicate the child is looking away. If a key is provided, then the trial will end if the child looks away looks long enough per the lookawayType and lookawayThreshold. You can also use 'mouse' to indicate that mouse down/up should be used in place of key down/up events. Use an empty string, '', to not record any lookaways for this trial. You can look up the names of keys at https://keycode.info.

endTrialKey [String | `'q'`] Key parent should press to manually move on to next trial. This allows you to have parents control the study by giving instructions like "press q when the child looks away for at least a few seconds" instead of "hold down w whenever the child isn't looking." Use an empty string, '', to not allow this function for this trial. You can look up the names of keys at https://keycode.info. Default is 'q'.

lookawayTone [String | `'noise'`] Type of audio to play during parent-coded lookaways - 'tone' (A 220), 'noise' (pink noise), or 'none'. These tones are available at https://www.mit.edu/~kimscott/placeholderstimuli/ if you want to use them in instructions.

lookawayToneVolume [Number | `0.25`] Volume of lookaway tone, as fraction of full volume (1 = full volume, 0 = silent)

4.2.3 Data collected

In addition to data collected by the regular version of the frame, a frame that uses this mixin will collect two pieces of data for convenience when coding or if implementing a live habituation procedure:

totalLookingTime [Number] Total looking time during this frame, in seconds. Looking time is calculated as the total time spent looking between:

1. The start of the parent control period, or the first look during that period if the child is not looking initially and

2. The end of the trial due to the parent pushing the end trial key, the child reaching the lookaway criterion, or the frame being completed without either of these happening (e.g., a video is played N times or an image is shown for N seconds).

All time spent looking away, per parent coding, is excluded, regardless of the duration of the lookaway.

This value will be null if the trial is not completed by any of the above mechanisms, for instance because the parent ends the study early during this frame.

trialEndReason [String] What caused the trial to end: 'lookaway' (the child reached the lookaway threshold), 'parentEnded' (the parent pressed the endTrialKey), or 'ceiling' (the frame ended without either of those happening).

This value will be null if the trial is not completed by any of the above mechanisms, for instance because the parent ends the study early during this frame.

4.2.4 Events recorded

In addition to events recorded by the regular version of the frame, a frame that uses this mixin will record the following events:

lookawayStart When parent records a lookaway starting. This will be triggered at the start of this frame if the parent is already holding down the lookawayKey, and otherwise only when the key is newly pressed down. Lookaways are recorded regardless of whether the parent control period has started.

lookawayEndedTrial When trial ends due to lookaway criterion being reached.

lookawayEnd When parent records a lookaway ending. This will NOT be triggered at the start of this frame if the parent is not holding down the lookawayKey, only when the key is actually released. Lookaways are recorded regardless of whether the parent control period has started.

parentControlPeriodStart When interval of parent control of trial begins - i.e., lookaways begin counting up to threshold. Lookaway events are recorded throughout, but do not count towards ending trial until parent control period begins.

parentEndedTrial When trial ends due to parent pressing key to end trial

parentControlPeriodEnd When interval of parent control of trial ends - i.e., lookaways cannot lead to ending trial, parent cannot press key to end trial.

4.3 video-record mixin

4.3.1 Overview

This mixin allows frames to display the webcam to the user and/or make recordings specific to the frame. For instance, *exp-lookit-video* can make a recording of the interval when the stimuli are displayed.

In general, the recorder is "installed" as soon as the frame loads. Once the recorder is installed, recording can begin quickly. Recording may begin right away (e.g. upon showing stimuli), upon the user pressing a button (e.g. in *exp-lookit-stimuli-preview*), or not at all (e.g. in *exp-video-config* where the webcam is displayed but not recorded).

Individual-frame and session-level recording

If there is an ongoing session-level recording, the camera will not be used by this frame and no recording will be made, to avoid overlapping recordings. This may interfere with frames such as *exp-lookit-webcam-display* that need to display the webcam.

If this occurs, a warning is displayed in the browser console.

Displaying a message during setup and/or upload

On frames that start and end recording automatically, you may want to hide stimuli and/or display an informative message during the second or two when the recorder is being set up and finishing video upload, to avoid displaying stimuli during time you won't have webcam video for.

You can set the following parameters to `true` independently:

- showWaitForRecordingMessage
- showWaitForUploadMessage

You can then customize what is displayed by setting the corresponding messages and their background colors. (The text color will be automatically set to either black or white to be maximally visible against the background.) You can additionally specify a single image or video to be shown during these periods.

4.3.2 Parameters

If your frame uses the video-record mixin, you can specify the following in addition to frame-specific parameters:

maxRecordingLength [Number|`7200`] Maximum recording length in seconds

maxUploadSeconds [Number|`5`] Maximum time allowed for video upload before proceeding to next frame, in seconds. Can be overridden by researcher, based on tradeoff between making families wait and losing data.

showWaitForRecordingMessage [Boolean|`true`] Whether to initially show a message saying to wait until recording starts, covering the entire frame. This prevents participants from seeing any stimuli before recording begins. Only used if recording is being started immediately.

waitForRecordingMessage [String|`'Please wait... <br><br> starting webcam recording'`] [Only used if showWaitForRecordingMessage is true] Text to display while waiting for recording to begin.

waitForRecordingMessageColor [String|`'white'`] [Only used if showWaitForRecordingMessage is true] Background color of screen while waiting for recording to begin. See https://developer.mozilla.org/en-US/docs/Web/CSS/color_value for acceptable syntax: can use either color names ('blue', 'red', 'green', etc.), or rgb hex values (e.g. '#800080' - include the '#'). The text on top of this will be either black or white depending on which will have higher contrast.

showWaitForUploadMessage [Boolean|`false`] Whether to stop media and hide stimuli with a message saying to wait for video upload when stopping recording. Do NOT set this to true if end of recording does not correspond to end of the frame (e.g. during consent or observation frames) since it will hide everything upon stopping the recording!

waitForUploadMessage [String|`'Please wait... <br><br> uploading video'`] [Only used if showWaitForUploadMessage is true] Text to display while waiting for recording to begin.

waitForUploadMessageColor [String|`'white'`] [Only used if showWaitForUploadMessage is true] Background color of screen while waiting for recording to upload. See https://developer.mozilla.org/en-US/docs/Web/CSS/color_value for acceptable syntax: can use either color names ('blue', 'red', 'green', etc.), or rgb hex values (e.g. '#800080' - include the '#'). The text on top of this will be either black or white depending on which will have higher contrast.

waitForWebcamImage [String] [Only used if showWaitForUploadMessage and/or showWaitForRecordingMessage are true] Image to display along with any wait-for-recording or wait-for-upload message. Either waitForWebcamImage or waitForWebcamVideo can be specified. This can be either a full URL ('https://...') or just a filename, which will be assumed to be inside `baseDir/img/` if this frame otherwise supports use of `baseDir`.

waitForWebcamVideo [String or Array] [Only used if showWaitForUploadMessage and/or showWaitForRecordingMessage are true] Video to display along with any wait-for-recording or wait-for-upload message (looping). Either waitForWebcamImage or waitForWebcamVideo can be specified. This can be either an array of `{'src': 'https://...', 'type': '...'}` objects (e.g. providing both webm and mp4 versions at specified URLS) or a single string relative to `baseDir/<EXT>/` if this frame otherwise supports use of `baseDir`.

4.3.3 Data collected

If your frame uses the video-record mixin, you will receive the following in addition to frame-specific data:

videoId The last video filename used during this frame (typically the only one). Format is *videoStream_<experimentId>_<frameId>_<sessionId>_timestampMS_RRR* where RRR are random numeric digits.

videoList A list of all video filenames created during this frame (a new one is created for each recording).

4.3.4 Events recorded

If your frame uses the video-record mixin, you may see the following in addition to frame-specific events:

hasCamAccess When recorder detects a change in camera access

>> **hasCamAccess** [Boolean] whether the recorder now has access

videoStreamConnection When recorder detects a change in video stream connection status

>> **status** [String] status status of video stream connection, e.g. 'NetConnection.Connect.Success' if successful

recorderReady When video recorder has been installed and can be started

startRecording When video recorder has actually started recording

>> **pipeId** [String] Original filename assigned by the Pipe client. May be used for troubleshooting.

stoppingCapture Just before stopping webcam video capture

destroyingRecorder When video recorder is about to be destroyed before next frame

RANDOMIZATION

Generally, you'll want to show slightly different versions of the study to different participants: perhaps you have a few different conditions, and/or need to counterbalance the order of trials or left/right position of stimuli. You have several options for how to handle this, depending on your preferences and the complexity of your design.

5.1 Randomization

5.1.1 Options for condition assignment and counterbalancing

Generally, you'll want to show slightly different versions of the study to different participants: perhaps you have a few different conditions, and/or need to counterbalance the order of trials or left/right position of stimuli. You have several options for how to handle this, depending on your preferences and the complexity of your design:

1. Simple randomization using frame parameters

A lot of simple randomization, like showing stimuli or questions in a random order, can be achieved simply using frame parameters to select values from a list you define. See *Frame parameters*.

2. Generating your study protocol using Javascript

You also have the option to provide a Javascript function that generates your study protocol programmatically. For complex counterbalancing designs, this may be simpler to reason about and debug than using randomizer frames (next) because you can define variables and write the step-by-step instructions for how to create the study protocol, without having to learn any special Lookit syntax. See *'Protocol generators'* for more information.

A protocol generator function can do anything that a randomizer frame can do. But to set up *conditional logic* (doing different things depending on what the family does *this session*), you will still need to use generateProperties or selectNextFrame parameters within the protocol you generate.

3. Randomizer frames

You can also use a special frame type called a **randomizer** to select an appropriate sequence of frames for a particular trial. A randomizer frame is automatically expanded to a list of frames, so that for instance you can specify your 12 looking-time trials all at once.

Randomizer frames allows you to randomize the study procedure without writing any code. These can be used to set up counterbalancing or condition assignment.

This section includes documentation about the randomizers *permute*, *random-parameter-set*, and *select*. To use a randomizer frame, set the frame `"kind"` to `"choice"` and `"sampler"` to the appropriate type of randomizer. The documentation will tell you how each randomizer works and what parameters you need to give it.

5.1.2 Case study: 2 x 2 x 2 design

Suppose you want to set up a study with a 2 x 2 x 2 design: that is, three types of things vary, each with two options. For this toy example, all we want to do is tell a short background story. The conditions will be:

- Character name: JANE or JILL

- Animal type: The character has a DOG or a CAT

- Location: The character lives in the COUNTRY or in the CITY

You want to create a single `exp-lookit-text` frame like this:

```
{
    "kind": "exp-lookit-text",
    "blocks": [
        {
            "text": "CHARACTER_INTRO_TEXT"
        },
        {
            "text": "ANIMAL_INTRO_TEXT"
        },
        {
            "text": "SETTING_TEXT"
        }
    ]
}
```

You have a variety of options for how to accomplish random condition assignment:

1. You could use a `random-parameter-set` randomizer and simply list all 2 * 2 * 2 = 8 options. Eight is a lot to list manually, but it's not ridiculous. This gives you maximum flexibility if you want to stop running one particular combination, or balance out the particular combinations based on how many kids in sub-age-ranges have completed each version of your study:

```
{
    "kind": "choice",
    "sampler": "random-parameter-set",
    "frameList": [
        {
            "kind": "exp-lookit-text",
            "blocks": [
                {
                    "text": "CHARACTER_INTRO_TEXT"
                },
```

```
                        {
                            "text": "ANIMAL_INTRO_TEXT"
                        },
                        {
                            "text": "SETTING_TEXT"
                        }
                    ]
                }
            ],
            "parameterSets": [
                {
                    "CHARACTER_INTRO_TEXT": "Once upon a time there was a girl named Jane.",
                    "ANIMAL_INTRO_TEXT": "She went everywhere with her dog.",
                    "SETTING_TEXT": "They lived in the middle of a big city."
                },
                {
                    "CHARACTER_INTRO_TEXT": "Once upon a time there was a girl named Jane.",
                    "ANIMAL_INTRO_TEXT": "She went everywhere with her dog.",
                    "SETTING_TEXT": "They lived out in the country."
                },
                {
                    "CHARACTER_INTRO_TEXT": "Once upon a time there was a girl named Jane.",
                    "ANIMAL_INTRO_TEXT": "She went everywhere with her cat.",
                    "SETTING_TEXT": "They lived in the middle of a big city."
                },
                {
                    "CHARACTER_INTRO_TEXT": "Once upon a time there was a girl named Jane.",
                    "ANIMAL_INTRO_TEXT": "She went everywhere with her cat.",
                    "SETTING_TEXT": "They lived out in the country."
                },
                {
                    "CHARACTER_INTRO_TEXT": "Once upon a time there was a girl named Jill.",
                    "ANIMAL_INTRO_TEXT": "She went everywhere with her dog.",
                    "SETTING_TEXT": "They lived in the middle of a big city."
                },
                {
                    "CHARACTER_INTRO_TEXT": "Once upon a time there was a girl named Jill.",
                    "ANIMAL_INTRO_TEXT": "She went everywhere with her dog.",
                    "SETTING_TEXT": "They lived out in the country."
                },
                {
                    "CHARACTER_INTRO_TEXT": "Once upon a time there was a girl named Jill.",
                    "ANIMAL_INTRO_TEXT": "She went everywhere with her cat.",
                    "SETTING_TEXT": "They lived in the middle of a big city."
                },
                {
                    "CHARACTER_INTRO_TEXT": "Once upon a time there was a girl named Jill.",
                    "ANIMAL_INTRO_TEXT": "She went everywhere with her cat.",
                    "SETTING_TEXT": "They lived out in the country."
                }
            ]
        }
```

2. If you don't want to deal with manually listing those combinations (for instance, because you're actually running a 2 x 2 x 2 x 2 x 2 design, or a 3 x 3 x 3 design...), you can use nested randomizers as discussed further below:

```json
{
    "kind": "choice",
    "sampler": "random-parameter-set",
    "frameList": [
        {
            "kind": "choice",
            "sampler": "random-parameter-set",
            "frameList": [
                {
                    "kind": "choice",
                    "sampler": "random-parameter-set",
                    "frameList": [
                        {
                            "kind": "exp-lookit-text",
                            "blocks": [
                                {
                                    "text": "CHARACTER_INTRO_TEXT"
                                },
                                {
                                    "text": "ANIMAL_INTRO_TEXT"
                                },
                                {
                                    "text": "SETTING_TEXT"
                                }
                            ]
                        }
                    ],
                    "parameterSets": [
                        {
                            "SETTING_TEXT": "They lived in the middle of a big city."
                        },
                        {
                            "SETTING_TEXT": "They lived out in the country."
                        }
                    ]
                }
            ],
            "parameterSets": [
                {
                    "ANIMAL_INTRO_TEXT": "She went everywhere with her cat."
                },
                {
                    "ANIMAL_INTRO_TEXT": "She went everywhere with her dog."
                }
            ]
        }
    ],
    "parameterSets": [
        {
            "CHARACTER_INTRO_TEXT": "Once upon a time there was a girl named Jane."
        },
        {
            "CHARACTER_INTRO_TEXT": "Once upon a time there was a girl named Jill."
        }
    ]
}
```

3. You can use the #RAND syntax and frame parameters to substitute in one of the two options for each condition:

```
{
    "kind": "exp-lookit-text",
    "blocks": [
        {
            "text": "CHARACTER_INTRO_TEXT_CHOICES#RAND"
        },
        {
            "text": "ANIMAL_INTRO_TEXT_CHOICES#RAND"
        },
        {
            "text": "SETTING_TEXT_CHOICES#RAND"
        }
    ],
    "parameters": {
        "CHARACTER_INTRO_TEXT_CHOICES": [
            "Once upon a time there was a girl named Jane.",
            "Once upon a time there was a girl named Jill."
        ],
        "ANIMAL_INTRO_TEXT_CHOICES": [
            "She went everywhere with her dog.",
            "She went everywhere with her cat."
        ],
        "SETTING_TEXT_CHOICES": [
            "They lived in the middle of a big city.",
            "They lived out in the country."
        ]
    }
}
```

Real randomization will generally be somewhat more complex - rather than setting the text on a single frame, you might be selecting which set of images to use, selecting whether to include a training phase, etc. However, the basic principles will be the same, and if you understand the options above, you will likely have a good idea of how to set up your own study.

5.1.3 Nested randomizers

In more complex experimental designs, the frames created by a randomizer may themselves be frame groups or randomizers! This nesting allows more modular specification: for instance, a study might have ten test trials, each of which consists of three phases. The "outer" randomizer could then generate a frameList of ten randomizer frames, each of which would be resolved in turn into three frames. Below is a simplified example with only two test trials, each of which has three phases:

Here's an example. Notice that `"kind": "choice"`, `"sampler": "random-parameter-set"`, `"frameList":  ...`, and `commonFrameProperties` are `commonFrameProperties` of the outer frame `nested-trials`. That means that every "frame" we'll create as part of `nested-trials` will itself be a random-parameter-set generated list with the same frame sequence, although we'll be substituting in different parameter values. (This doesn't have to be the case - we could show different types of frames in the list - but in the simplest case where you're using randomParameterSet just to group similar repeated frame sequences, this is probably what you'd do.) The only thing that differs across the two (outer-level) **trials** is the `parameterSet` used, and we list only one parameter set for each trial, to describe (deterministically) how the outer-level `parameterSet` values should be applied to each particular frame.

```
{
    "sampler": "random-parameter-set",
    "frameList": [
```

(continues on next page)

```
    {
      "parameterSets": [
          {
            "NTRIAL": 1,
        "PHASE1STIM": "T1P1",
        "PHASE2STIM": "T1P2",
        "PHASE3STIM": "T1P3"
          }
      ]
    },
    {
      "parameterSets": [
          {
            "NTRIAL": 2,
        "PHASE1STIM": "T2P1",
        "PHASE2STIM": "T2P2",
        "PHASE3STIM": "T2P3"
          }
      ]
    }
  ],
  "parameterSets": [
    {
        "T1P1": "mouse",
        "T1P2": "rat",
        "T1P3": "chipmunk",
        "T2P1": "horse",
        "T2P2": "goat",
        "T2P3": "cow"
    },
    {
        "T1P1": "guppy",
        "T1P2": "tadpole",
        "T1P3": "goldfish",
        "T2P1": "whale",
        "T2P2": "manatee",
        "T2P3": "shark"
    }

  ],
  "commonFrameProperties": {
    "sampler": "random-parameter-set",
    "frameList": [
          {
              "nPhase": 1,
            "animal": "PHASE1STIM"
            },
            {
              "nPhase": 2,
            "animal": "PHASE2STIM"
            },
            {
              "nPhase": 3,
            "animal": "PHASE3STIM"
            }
      ],
    "commonFrameProperties": {
```

```
                "nTrial": "NTRIAL",
                "kind": "question-about-animals-frame"
            }
        }
    }
}
```

To evaluate this experiment frame, the Lookit experiment player starts with the list of frames in the outer `frameList`, adding the key:value pairs in the outer `commonFrameProperties` to each frame, which yields the following list of frames:

```
[
        {
        "parameterSets": [
                {
                    "NTRIAL": 1,
                "PHASE1STIM": "T1P1",
                "PHASE2STIM": "T1P2",
                "PHASE3STIM": "T1P3"
            }
        ],
        "sampler": "random-parameter-set",
        "frameList": [
            {
                "nPhase": 1,
            "animal": "PHASE1STIM"
            },
            {
                "nPhase": 2,
            "animal": "PHASE2STIM"
            },
            {
                "nPhase": 3,
            "animal": "PHASE3STIM"
            }
        ],
        "commonFrameProperties": {
            "nTrial": "NTRIAL",
            "kind": "question-about-animals-frame"
        }
        },
        {
            "parameterSets": [
                {
                    "NTRIAL": 2,
                "PHASE1STIM": "T2P1",
                "PHASE2STIM": "T2P2",
                "PHASE3STIM": "T2P3"
                }
            ],
        "sampler": "random-parameter-set",
        "frameList": [
            {
                "nPhase": 1,
            "animal": "PHASE1STIM"
            },
            {
```

5.1. Randomization

135

```
                "nPhase": 2,
            "animal": "PHASE2STIM"
            },
            {
                "nPhase": 3,
            "animal": "PHASE3STIM"
            }
        ],
        "commonFrameProperties": {
            "nTrial": "NTRIAL",
            "kind": "question-about-animals-frame"
        }
    }
]
```

One of the two (outer) `parameterSets` is then selected randomly; suppose the second one (aquatic instead of land animals) is selected. Now any substitutions are made based on the keys in this parameterSet. The first frame in the sequence is now:

```
{
    "parameterSets": [
            {
                "NTRIAL": 1,
            "PHASE1STIM": "guppy",
            "PHASE2STIM": "tadpole",
            "PHASE3STIM": "goldfish"
        }
    ],
    "sampler": "random-parameter-set",
    "frameList": [
        {
            "nPhase": 1,
        "animal": "PHASE1STIM"
        },
        {
            "nPhase": 2,
        "animal": "PHASE2STIM"
        },
        {
            "nPhase": 3,
        "animal": "PHASE3STIM"
        }
    ],
    "commonFrameProperties": {
        "nTrial": "NTRIAL",
        "kind": "question-about-animals-frame"
    }
}
```

Next, each frame is expanded since it is in turn another randomizer (due to `"sampler":` `"random-parameter-set"`). The frame above, representing Trial 1, will be turned into three frames. First, again, we start with the `frameList`, and merge the `commonFrameProperties` into each frame:

```
[
    {
        "nPhase": 1,
```

```
            "animal": "PHASE1STIM",
            "nTrial": "NTRIAL",
                "kind": "question-about-animals-frame"
        },
        {
            "nPhase": 2,
            "animal": "PHASE2STIM",
            "nTrial": "NTRIAL",
                "kind": "question-about-animals-frame"
        },
        {
            "nPhase": 3,
            "animal": "PHASE3STIM",
            "nTrial": "NTRIAL",
                "kind": "question-about-animals-frame"
        }
]
```

Finally, a parameter set is selected from `parameterSets`. Only one parameter set is defined for this trial, which is deliberate; it simply selects the correct stimuli for this trial. Substituting in the values from the parameter set yields the following list of frames:

```
[
        {
            "nPhase": 1,
            "animal": "guppy",
            "nTrial": 1,
                "kind": "question-about-animals-frame"
        },
        {
            "nPhase": 2,
            "animal": "tadpole",
            "nTrial": 1,
                "kind": "question-about-animals-frame"
        },
        {
            "nPhase": 3,
            "animal": "goldfish",
            "nTrial": 1,
                "kind": "question-about-animals-frame"
        }
]
```

The `random-parameter-set` randomizer is expected to be general enough to capture most experimental designs that researchers put on Lookit, but additional more specific randomizers will also be designed to provide simpler syntax for common use cases.

5.2 permute

5.2.1 Overview

Randomizer to allow random ordering of a list of frames. Intended to be useful for e.g. randomly permuting the order of particular stimuli or blocks of trials. Frames don't need to be of the same kind to permute.

To use, define a frame with "kind": "choice" and "sampler": "permute", as shown below, in addition to the parameters described below.

5.2.2 Example

Half the time, this will show "Let's think about hippos!" first, followed by "Let's think about dolphins!" The other half, this will show "Let's think about dolphins!" first.

```
"test-trials": {
    "sampler": "permute",
    "kind": "choice",
    "commonFrameProperties": {
        "showPreviousButton": false
    },
    "frameOptions": [
        {
            "blocks": [
                {
                    "emph": true,
                    "text": "Let's think about hippos!",
                    "title": "hippos!"
                },
                {
                    "text": "Some more about hippos..."
                }
            ],
            "kind": "exp-lookit-text"
        },
        {
            "blocks": [
                {
                    "emph": false,
                    "text": "Let's think about dolphins!",
                    "title": "dolphins!"
                }
            ],
            "kind": "exp-lookit-text"
        }
    ]
}
```

5.2.3 Parameters

frameOptions [Array] List of frames to be created by this randomizer. Each frame is an object with any necessary frame-specific properties specified. The 'kind' of frame can be specified either here (per frame) or in commonFrameProperties. If a property is defined for a given frame both in this frame list and in common-FrameProperties, the value in the frame list will take precedence.

(E.g., you could include `'kind':  'normal-frame'` in commmonFrameProperties, but for a single frame in frameOptions, include `'kind':  'special-frame'`.)

orderedFrameOptions [Array] List of objects containing frame properties. The list should be the same length as frameOptions. The properties in the first element of this list will be added to the frame shown first, the properties in the second element of this list will be added to the frame shown second, and so on.

This allows you to, for instance, do something different during the first or last trial (e.g., treating the first three trials as practice/training), or provide audio indicating progress through the study based on which trial number you're on - even though you're shuffling the order with permute!

If `parameterSets` is included as one of the properties in `orderedFrameOptions[n]`, the values will be *added* to any parameterSets property on the existing frame (value-by-value, iterating through corresponding parameterSets) rather than overwriting the whole property.

commonFrameProperties [Object | {}] Object describing common parameters to use as defaults for every frame created by this randomizer. Parameter names and values are as described in the documentation for the frameType used.

5.2.4 Data collected

The information returned by this randomizer will be available in `expData["conditions"]["THIS-RANDOMIZER-ID"]`. The randomizer ID will depend on its order in the study - for instance, `6-test-trials`.

frameList [Array] the list of frames used, in the final shuffled order

5.3 random-parameter-set

5.3.1 Overview

Randomizer that provides flexible condition assignment and counterbalancing by allowing the user to specify an arbitrary sequence of frames to create. A set of parameters is randomly selected from a list of available `parameterSets`, and these parameters are substituted in to the parameters specified in the list of frames.

To use, define a frame with 'kind': 'choice' and 'sampler': 'random-parameter-set', as shown below, in addition to the parameters described under 'properties'.

```
{
    ...
    "frames": {
        ...
        "test-trials": {
            "sampler": "random-parameter-set",
            "kind": "choice",
            ...
        }
    }
}
```

In addition, there are two special properties you need to define to use the `random-parameter-set` randomizer:

- `frameList`: this is just what it sounds like: a list of all the frames that should be generated by this randomizer. Each frame is a JSON object just like you would use in the overall schema, with two differences:

  - You can define default properties, to share across all of the frames generated by this randomizer, in the JSON object `commonFrameProperties` instead, as a convenience.

  - You can use placeholder strings for any of the properties in the frame; they will be replaced based on the values in the selected `parameterSet`.

- `parameterSets` is a list of mappings from placeholder strings to actual values. When a participant starts your study, one of these sets will be randomly selected, and any parameter values in the `frameList` (including `commonFrameProperties`) that match any of the keys in this parameter set will be replaced.

Advanced options for choosing the parameterSet

You can determine the weights based on the child's age, to maintain balanced conditions.) You can also keep kids in the same condition across all sessions they complete, or rotate them through conditions in order.

Example

Let's walk through an example of using this randomizer. Suppose we start with the following JSON document describing a study that includes instructions, an experimental manipulation asking participants to think about how delicious broccoli is, and an exit survey:

```
{
    "frames": {
        "instructions": {
            "id": "text-1",
            "blocks": [
                {
                    "text": "Some introductory text about this study."
                },
                {
                    "text": "Here's what's going to happen! You're going to think␣
↪about how tasty broccoli is."
                }
            ],
            "showPreviousButton": false,
            "kind": "exp-lookit-text"
        },
        "manipulation": {
            "id": "text-2",
            "blocks": [
                {
                    "text": "Think about how delicious broccoli is."
                },
                {
                    "text": "It is so tasty!"
                }
            ],
            "showPreviousButton": true,
            "kind": "exp-lookit-text"
        },
        "exit-survey": {
```

(continues on next page)

```
            "debriefing": {
                "text": "Thank you for participating in this study! ",
                "title": "Thank you!"
            },
            "id": "exit-survey",
            "kind": "exp-lookit-exit-survey"
        }
    },
    "sequence": [
        "instructions",
        "manipulation",
        "exit-survey"
    ]
}
```

But what we really want to do is have some kids think about how tasty broccoli is, and others think about how yucky it is! We can use a `random-parameter-set` frame to replace both text frames:

```
{
    "frames": {
        "instruct-and-manip": {
            "sampler": "random-parameter-set",
            "kind": "choice",
            "id": "instruct-and-manip",
            "frameList": [
                {
                    "blocks": [
                        {
                            "text": "Some introductory text about this study."
                        },
                        {
                            "text": "INTROTEXT"
                        }
                    ],
                    "showPreviousButton": false
                },
                {
                    "blocks": [
                        {
                            "text": "MANIP-TEXT-1"
                        },
                        {
                            "text": "MANIP-TEXT-2"
                        }
                    ],
                    "showPreviousButton": true
                }
            ],
            "commonFrameProperties": {
                "kind": "exp-lookit-text"
            },
            "parameterSets": [
                {
                    "INTROTEXT": "Here's what's going to happen! You're going to
→think about how tasty broccoli is.",
                    "MANIP-TEXT-1": "Think about how delicious broccoli is.",
```

```
                "MANIP-TEXT-2": "It is so tasty!"
            },
            {
                "INTROTEXT": "Here's what's going to happen! You're going to␣
→think about how disgusting broccoli is.",
                "MANIP-TEXT-1": "Think about how disgusting broccoli is.",
                "MANIP-TEXT-2": "It is so yucky!"
            }
        ]
    },
    "exit-survey": {
        "debriefing": {
            "text": "Thank you for participating in this study! ",
            "title": "Thank you!"
        },
        "id": "exit-survey",
        "kind": "exp-lookit-exit-survey"
    }
},
"sequence": [
    "instruct-and-manip",
    "exit-survey"
]
}
```

Notice that since both of the frames in the `frameList` were of the same kind, we could define the kind in `commonFrameProperties`. We no longer define `id` values for the frames, as they will be automatically identified as `instruct-and-manip-1` and `instruct-and-manip-2`.

When the "instruct-and-manip" randomizer is evaluated, the Lookit experiment player will start with the frameList and add the key-value pairs in commonFrameProperties to each frame (not overwriting existing pairs):

```
[
    {
        "kind": "exp-lookit-text",
        "blocks": [
            {
                "text": "Some introductory text about this study."
            },
            {
                "text": "INTROTEXT"
            }
        ],
        "showPreviousButton": false
    },
    {
        "kind": "exp-lookit-text",
        "blocks": [
            {
                "text": "MANIP-TEXT-1"
            },
            {
                "text": "MANIP-TEXT-2"
            }
        ],
        "showPreviousButton": true
    }
```

```
]
```

Next, one of the two objects in `parameterSets` is selected randomly. (By default, parameter sets are weighted equally, but `parameterSetWeights` can be provided as an optional key in the `random-parameter-set` frame. If provided, `parameterSetWeights` should be an array of relative weights for the parameter sets, corresponding to the order they are listed. For instance, if we wanted 75% of participants to think about how tasty broccoli is, we could set `parameterSetWeights` to [3, 1]. This allows uneven condition assignment where needed to optimize power, as well as allowing researchers to stop testing conditions that already have enough participants as data collection proceeds.

Suppose that in this case the second parameter set is selected:

```
{
"INTROTEXT": "Here's what's going to happen! You're going to think about how␣
↪disgusting broccoli is.",
"MANIP-TEXT-1": "Think about how disgusting broccoli is.",
"MANIP-TEXT-2": "It is so yucky!"
}
```

Now we return to the list of frames, and wherever any value matches one of the keys in the `parameterSet` (even if that value is nested in another object), it is replaced by the corresponding value from the `parameterSet`, yielding the following final list of frames:

```
[
    {
        "kind": "exp-lookit-text",
        "blocks": [
            {
                "text": "Some introductory text about this study."
            },
            {
                "text": "Here's what's going to happen! You're going to think about␣
↪how disgusting broccoli is."
            }
        ],
        "showPreviousButton": false
    },
    {
        "kind": "exp-lookit-text",
        "blocks": [
            {
                "text": "Think about how disgusting broccoli is."
            },
            {
                "text": "It is so yucky!"
            }
        ],
        "showPreviousButton": true
    }
]
```

5.3.2 Parameters

commonFrameProperties [Object] Object describing common parameters to use in EVERY frame created by this randomizer. Parameter names and values are as described in the documentation for the frameType used.

frameList [Array] List of frames to be created by this randomizer. Each frame is an object with any necessary frame-specific properties specified. The `kind` of frame can be specified either here (per frame) or in `commonFrameProperties`. If a property is defined for a given frame both in this frame list and in `commonFrameProperties`, the value in the frame list will take precedence.

(E.g., you could include `'kind': 'normal-frame'` in `commonFrameProperties`, but for a single frame in `frameList`, include `'kind': 'special-frame'`.)

Any property *values* within any of the frames in this list which match a property *name* in the selected `parameterSet` will be replaced by the corresponding `parameterSet` value. For example, suppose a frame in `frameList` is

```
{
    'leftImage': 'LEFTIMAGE1',
    'rightImage': 'frog.jpg',
    'size': 'IMAGESIZE'
}
```

and the row that has been selected randomly of *parameterSets* is

```
{
    'LEFTIMAGE1': 'toad.jpg',
    'LEFTIMAGE2': 'dog.jpg',
    'IMAGESIZE': 250
}
```

Then the frame would be transformed into:

```
{
    'leftImage': 'toad.jpg',
    'rightImage': 'frog.jpg',
    'size': 250
}
```

The same values may be applied across multiple frames. For instance, suppose *frameList* is

```
[
    {
        'leftImage': 'LEFTIMAGE1',
        'rightImage': 'frog.jpg',
        'size': 'IMAGESIZE'
    },
    {
        'leftImage': 'LEFTIMAGE2',
        'rightImage': 'frog.jpg',
        'size': 'IMAGESIZE'
    }
]
```

Then the corresponding processed frames would include the values

```
[
    {
```

```
        'leftImage': 'toad.jpg',
        'rightImage': 'frog.jpg',
        'size': 250
    },
    {

        'leftImage': 'dog.jpg',
        'rightImage': 'frog.jpg',
        'size': 250

    }
]
```

A property value like `IMAGESIZE` may be placed in a frame definition nested within another object (at any depth) or within a list and will still be replaced.

You can also use selectors to randomly sample from or permute a list given in a `parameterSet`. Suppose `LISTVAR` is defined in a `parameterSet` as `THELIST`, e.g. a list of potential stimuli. Within frames in your `frameList` (and in `commonFrameProperties`), you can use any of the following:

- Select the Nth element (0-indexed) of `THELIST`: (Will cause error if `N >= THELIST.length`)

```
'parameterName': 'LISTVAR#N'
```

- Select (uniformly) a random element of `THELIST`:

```
'parameterName': 'LISTVAR#RAND'
```

- Set `parameterName` to a random permutation of `THELIST`:

```
'parameterName': 'LISTVAR#PERM'
```

- Select the next element in a random permutation of `THELIST`, which is used across all substitutions in this randomizer. This allows you, for instance, to provide a list of possible images in your `parameterSet`, and use a different one each frame with the subset/order randomized per participant. If more `LISTVAR_UNIQ` parameters than elements of `THELIST` are used, we loop back around to the start of the permutation generated for this randomizer.

```
'parameterName': 'LISTVAR#UNIQ'
```

parameterSets [Array] Array of parameter sets to randomly select from in order to determine the parameters for each frame in this session.

A single element of *parameterSets* will be applied to a given session.

conditionForAdditionalSessions [String | `'random'`]

[Optional] How to select a parameterSet for a participant who has previously participated in this study. Must be one of `'random'` (default), `'persist'`, or `'rotate'`. Meanings:

- `random`: regardless of any previous sessions from this participant, select a parameterSet for this participant as usual (including using parameterSetWeights if provided). Default behavior.

- `persist`: Continue assigning the same participant to the same parameterSet for all sessions.

- `rotate`: The first time, assign parameterSet randomly (per parameterSetWeights if given); after that, each time the participant participates assign them to the next parameterSet in the list. Subtracts length of parameterSets until the 'next' index is in range.

The most recent session in which the `conditions` data includes an element that looks like it was generated by this same randomizer (i.e., with key ending in `-frameId`, like `-test-trials`) will always be used for assignment.

Only sessions with a completed consent frame are considered, so that participants are not rotated through conditions simply due to refreshing the setup page.

The "same" or "next" parameterSets are determined by the **index** of the previously-selected parameterSet. That is, if you were assigned to conditionNum 0 (index 0 in parameterSets) last time, you will be assigned to conditionNum 0 again this time if `conditionForAdditionalSessions` is `"persist"` and conditionNum 1 if `conditionForAdditionalSessions` is `"rotate"`. So if you update the list of `parameterSets` in your study - e.g. to fix a bug or clarify wording - the new values will be used even for repeat participants. But be careful that you do not reorder them unless you intend to, say, swap all participants to the opposite condition on a specified date!

If the previous index is now outside the range of the `parameterSets` list (e.g., you used to have 6 conditions, and the participant was previously in condition number 5, but then you changed `parameterSets` to have only 3 elements) and `conditionForAdditionalSessions` is `"persist"`, then the participant is assigned to the last element of `parameterSets`.

parameterSetWeights [Array]

[Optional] Array of weights for parameter sets; elements correspond to elements of parameterSets. The probability of selecting an element `parameterSets[i]` is `parameterSetWeights[i]/ sum(parameterSetWeights)`.

If not provided, all `parameterSets` are weighted equally.

This is intended to allow manual control of counterbalancing during data collection, e.g. to allow one condition to 'catch up' if it was randomly selected less often.

Instead of providing a single list of the same length as parameterSets, you may instead provide a list of objects specifying the weights to use within various age ranges, like this:

```
'parameterSetWeights': [
    {
        'minAge': 0,
        'maxAge': 365,
        'weights': [1, 0, 1]
    },
    {
        'minAge': 365,
        'maxAge': 10000,
        'weights': [0, 1, 0]
    },
]
```

The child's age in days will be computed, and the weights used will be based on the first element of `parameterSetWeights` where the child falls between the min and max age. In the example above, children under one year old will be assigned to either the first or third condition; children over a year will be assigned to the second condition. This may be useful for researchers who need to balance condition assignment per age bracket. As you code data and realize you are set on 3-year-olds in condition A, for instance, you can stop assigning any more 3-year-olds to that condition.

5.3.3 Data collected

The information returned by this randomizer will be available in `expData["conditions"]["THIS-RANDOMIZER-ID"]`. The randomizer ID will depend on its order in the study - for instance, `6-test-trials`.

conditionNum [Number] the index of the parameterSet chosen

parameterSet [Object] the parameterSet chosen

5.4 select

5.4.1 Overview

Randomizer to allow selection of one (or arbitrary sequence) of defined frames.

This is unlikely to be useful on its own! It is intended to be used either:

- within a *random-parameter-set* randomizer, with different `parameterSets` picking out different values for `whichFrames`)

- via the *exp-frame-select* frame, which allows you to set *whichFrames* using a custom `generateProperties` function. (Note that you can't add a `generateProperties` function directly to a randomizer - that's why you'd use the regular frame instead.)

To use, define a frame with `"kind": "choice"` and `"sampler": "select"`, as shown below, in addition to the parameters described under 'properties'.

5.4.2 Example

This will always show "Let's think about hippos!" because it picks out the first frame (`"whichFrames": 0`):

```
"select-randomizer-test": {
    "sampler": "select",
    "kind": "choice",
    "whichFrames": 0,
    "commonFrameProperties": {
        "kind": "exp-lookit-text"
    },
    "frameOptions": [
        {
            "blocks": [
                {
                    "emph": true,
                    "text": "Let's think about hippos!",
                    "title": "FRAME 1"
                },
                {
                    "text": "Some more about hippos..."
                }
            ]
        },
        {
            "blocks": [
                {
                    "emph": false,
```

```
                "text": "Let's think about dolphins!",
                "title": "FRAME 2"
            }
        ]
    }
]
}
```

5.4.3 Parameters

frameOptions [Array] List of frames that can be created by this randomizer. Each frame is an object with any necessary frame-specific properties specified. The 'kind' of frame can be specified either here (per frame) or in commonFrameProperties. If a property is defined for a given frame both in this frame list and in commonFrameProperties, the value in the frame list will take precedence.

(E.g., you could include 'kind': 'normal-frame' in commmonFrameProperties, but for a single frame in frameOptions, include 'kind': 'special-frame'.)

commonFrameProperties [Object] Object describing common parameters to use in EVERY frame created by this randomizer. Parameter names and values are as described in the documentation for the frameType used.

whichFrames [Number or Array] Index or indices (0-indexed) within `frameOptions` to actually use. This can be either a number (e.g., 0 to use the first option or 1 to use the second option) or an array providing an ordered list of indices to use (e.g., [1, 0] to use the second then first options). All indices must be integers in [0, frameOptions.length).

If not provided or -1, the entire `frameOptions` list is used in order. (If an empty list is provided, however, that is respected and no frames are inserted by this randomizer.)

5.4.4 Data collected

The information returned by this randomizer will be available in `expData["conditions"]["THIS-RANDOMIZER-ID"]`. The randomizer ID will depend on its order in the study - for instance, `6-test-trials`.

whichFrames [Array] the index/indices of the frame(s) used, as provided to this frame

5.5 exp-text-block

5.5.1 Overview

This is a small utility for displaying text that a variety of frames use. If the frame documentation said that some parameter would be rendered by exp-text-block, or that it should be a list of blocks each to be rendered by exp-text-block, you're in the right place to learn how to format that parameter.

5.5.2 Examples

Single block: title and text

```
"example-text-frame": {
    "kind": "exp-lookit-text",
    "blocks": [
        {
            "title": "Lorem ipsum",
            "text": "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
↪eiusmod tempor incididunt ut labore et dolore magna aliqua. Suscipit adipiscing
↪bibendum est ultricies integer quis auctor. Imperdiet sed euismod nisi porta lorem
↪mollis. Sollicitudin tempor id eu nisl nunc mi. Aliquet lectus proin nibh nisl
↪condimentum. Ac tincidunt vitae semper quis lectus nulla at volutpat. Mauris sit
↪amet massa vitae tortor condimentum lacinia. Tincidunt vitae semper quis lectus
↪nulla at volutpat. Elementum curabitur vitae nunc sed. Pharetra convallis posuere
↪morbi leo."
        }
    ]
}
```

Lorem ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Suscipit adipiscing bibendum est ultricies integer quis auctor. Imperdiet sed euismod nisi porta lorem mollis. Sollicitudin tempor id eu nisl nunc mi. Aliquet lectus proin nibh nisl condimentum. Ac tincidunt vitae semper quis lectus nulla at volutpat. Mauris sit amet massa vitae tortor condimentum lacinia. Tincidunt vitae semper quis lectus nulla at volutpat. Elementum curabitur vitae nunc sed. Pharetra convallis posuere morbi leo.

Single block: text and image

```
"example-text-frame": {
    "kind": "exp-lookit-text",
    "blocks": [
        {
            "text": "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
↪eiusmod tempor incididunt ut labore et dolore magna aliqua. Suscipit adipiscing
↪bibendum est ultricies integer quis auctor. Imperdiet sed euismod nisi porta lorem
↪mollis. Sollicitudin tempor id eu nisl nunc mi. Aliquet lectus proin nibh nisl
↪condimentum. Ac tincidunt vitae semper quis lectus nulla at volutpat.",
            "image": {
                "src": "https://www.mit.edu/~kimscott/placeholderstimuli/img/apple.jpg
↪",
                "alt": "Red apple"
            }
        }
    ]
}
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Suscipit adipiscing bibendum est ultricies integer quis auctor. Imperdiet sed euismod nisi porta lorem mollis. Sollicitudin tempor id eu nisl nunc mi. Aliquet lectus proin nibh nisl condimentum. Ac tincidunt vitae semper quis lectus nulla at volutpat.

Note that there may be formatting applied to the image based on the particular frame.

Single block: text and listblocks

```
"example-text-frame": {
    "kind": "exp-lookit-text",
    "blocks": [
        {
            "text": "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
→eiusmod tempor incididunt ut labore et dolore magna aliqua.",
            "listblocks": [
                {
                    "text": "Suscipit adipiscing bibendum est ultricies integer quis
→auctor."
                },
                {
                    "text": "Imperdiet sed euismod nisi porta lorem mollis."
                },
                {
                    "text": "Sollicitudin tempor id eu nisl nunc mi."
                }
            ]
        }
    ]
}
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

- Suscipit adipiscing bibendum est ultricies integer quis auctor.

- Imperdiet sed euismod nisi porta lorem mollis.

- Sollicitudin tempor id eu nisl nunc mi.

Note that there may be formatting applied to the image based on the particular frame.

Three blocks

In many frames you can specify a list of blocks to be rendered by exp-text-block - here's an example:

```
"example-text-frame": {
    "kind": "exp-lookit-text",
    "blocks": [
        {
            "title": "Lorem ipsum",
            "text": "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do␣
→eiusmod tempor incididunt ut labore et dolore magna aliqua.",
            "listblocks": [
                {
                    "text": "Suscipit adipiscing bibendum est ultricies integer quis␣
→auctor."
                },
                {
                    "text": "Imperdiet sed euismod nisi porta lorem mollis."
                },
                {
                    "text": "Sollicitudin tempor id eu nisl nunc mi."
                }
            ]
        },
        {
            "title": "Nulla porttitor",
            "text": " Nulla porttitor massa id neque aliquam. Ac felis donec et odio␣
→pellentesque diam. Nisl vel pretium lectus quam id leo. "
        },
        {
            "image": {
                "src": "https://www.mit.edu/~kimscott/placeholderstimuli/img/apple.jpg
→",
                "alt": "Red apple"
            },
            "listblocks": [
                {
                    "text": "Est ante in nibh mauris cursus."
                },
                {
                    "text": "Ut aliquam purus sit amet luctus venenatis lectus."
                },
```

(continues on next page)

```
                        {
                            "text": "Cras ornare arcu dui vivamus arcu felis bibendum ut."
                        }
                    ]
                }
            ]
        }
```

Lorem ipsum

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

- Suscipit adipiscing bibendum est ultricies integer quis auctor.

- Imperdiet sed euismod nisi porta lorem mollis.

- Sollicitudin tempor id eu nisl nunc mi.

Nulla porttitor

Nulla porttitor massa id neque aliquam. Ac felis donec et odio pellentesque diam. Nisl vel pretium lectus quam id leo.

- Est ante in nibh mauris cursus.

- Ut aliquam purus sit amet luctus venenatis lectus.

- Cras ornare arcu dui vivamus arcu felis bibendum ut.

Inserting a link

```
"example-text-frame": {
    "kind": "exp-lookit-text",
    "blocks": [
        {
            "title": "Lorem ipsum",
            "text": "Here is a link to <a href='https://lookit.mit.edu/' target='_
→blank' rel='noopener'>Lookit</a>."
        }
    ]
}
```

5.5.3 Parameters

All of the following are optional:

title [String] Title text to display at the top of this block

text [String] Main text of this block. You can use *n* or *
* for paragraph breaks. You can use HTML inside the text, for instance to include a link or an image.

> **emph** whether to show this text in bold

image [Object] Image to display along with this block. Needs two fields:

> **src** URL of image
>
> **alt** alt-text for image

listblocks [Array] A list of items to display in bullet points. Each item is itself rendered with exp-text-block, so it is an object that can have title, text, image, listblocks, etc.

5.6 exp-lookit-composite-video-trial

This frame is deprecated and will not be included in release 2.x. Please see the old docs.

5.6.1 Consider instead

- *exp-lookit-video* to play a video and optional audio, with more configurability & better timing precision for the recording

- *exp-lookit-calibration* to show a calibration stimulus at specific locations, again with more configurability and better timing precision

5.7 exp-lookit-dialogue-page

This frame is deprecated and will not be included in release 2.x. Please see the old docs.

5.7.1 Consider instead

- *exp-lookit-images-audio*, which should be able to do everything this frame can, but more reliably and with more options!

5.8 exp-lookit-geometry-alternation

This frame is deprecated and will not be included in release 2.x. Please see the old docs.

5.8.1 Consider instead

The more general-purpose *exp-lookit-change-detection* frame which shows streams of images on each side, or *exp-lookit-video* if you'd rather provide premade videos.

5.9 exp-lookit-geometry-alternation-open

This frame is deprecated and will not be included in release 2.x. Please see the old docs.

5.9.1 Consider instead

The more general-purpose *exp-lookit-change-detection* frame which shows streams of images on each side, or *exp-lookit-video* if you'd rather provide premade videos.

5.10 exp-lookit-preferential-looking

This frame is deprecated and will not be included in release 2.x. Please see the old docs.

5.10.1 Consider instead

- *exp-lookit-video* to play a video and optional audio, with more configurability & better timing precision for the recording

- *exp-lookit-calibration* to show a calibration stimulus at specific locations, again with more configurability and better timing precision

5.11 exp-lookit-story-page

This frame is deprecated and will not be included in release 2.x. Please see the old docs.

5.11.1 Consider instead

- *exp-lookit-images-audio*, which should be able to do everything this frame can, but more reliably and with more options!