

Conversation Examples

The following code shows basic use of the Conversation module. The HTML on which it operates is assumed and not shown. jQuery is not required for the conversation module, however it's use in the examples provides simpler demonstration code.

```
// Always start with a new Conversation
let conv = new Conversation();
// Ralph and Sara are participants in this conversation.
// Ralph's speech will appear as red text in selector '#speech'
// Sara 's speech will appear as blue text in the same selector
let ralph = conv.defineParticipant('ralph', '#speech', { color: red; });
let sara = conv.defineParticipant('sara', '#speech', { color: blue; });

// their brief conversation follows (order is important)
conv.addStage(ralph, "Hello Sara");
conv.addStage(sara, "Hello Ralph");
conv.addStage(sara, "Do you like Superman?");
conv.addStage(ralph, "Sorry, I have to go now.");

// make the first conversation text visible immediately
conv.start();

// jQuery is not required for the conversation,
// but it makes for easier and clearer demonstration code
$('#button').click(function() {
  conv.next();
})
```

In this example two participants each have their own text-bubbles for speech. If the text bubbles start hidden (either directly or through an ancestor being hidden) they will be individually shown when required, and restored to hidden afterwards. (Allowing an ancestor to be hidden means a fancy text bubble with text in a child element will still work).

```
let conv = new Conversation();
let ralph = conv.defineParticipant('ralph', '#ralphSpeechBubble');
let sara = conv.defineParticipant('sara', '#saraSpeechBubble');

conv.addStage(ralph, "Hello <em>Sara</em>");
conv.addStage(sara, "Hello <em>Ralph</em>");
conv.label('ask age');
conv.addStage(ralph, "What is your age Sara?");
conv.addStage(sara, "A woman never tells her age");
conv.addStage(ralph, "Damn! Maybe if I ask again?");
// this time we don't call start(), so the first conversation won't
// be visible until the first click of the button #next
```

```
$('#button#next').click(function() {  
    conv.next();  
});  
$('#button#ask-age').click(function() {  
    conv.gotoLabel('ask age');  
});
```

Conversation API

v 0.1 24 Sept 2024

`new Conversation();`

This creates a conversation object, which should be used for all future calls. It takes no arguments. Unless otherwise mentioned, all methods listed below should be called on this conversation object.

`defineParticipant();`

This creates and returns a conversation participant. The returned value should be used as the first argument when adding conversation stages (see `addStage()`).

There are several forms which this method can take:

`defineParticipant( name, textSelector );`

`name` — a conversation participant's name (string). This may be anything but should be unique. It is primarily used for identification and debugging purposes. eg. "fred".

`textSelector` — a string as a selector, to identify a unique DOM element where the txt should be placed when *this* participant speaks. For example it could identify a speech bubble specific to this participant, or a common area where all speech should be shown. The behaviour for providing a non-unique selector is undefined.

`defineParticipant( name, textSelector, style );`

This is the same as `defineParticipant( name, textSelector )` with an additional styling object containing CSS key/value pairs. When this participant speaks, the provided style will be applied to the given `textSelector`. This can be useful when two or more participants share the same text selector, allowing different styles to be applied for each participant, indicating who is speaking.

Detail: the styling is set only when the participant changes. If the same person says two lines in a row (two stages) then the styles are not applied twice. Also styling is not removed at the end of the conversation. To remove styling at the end of a conversation assign a callback function to the `end` property of the conversation object. See property '`end`' documented below.

`defineParticipant(name, textSelector, style, talkStartFn, talkEndFn);`

This variation adds provision for two callback functions (which should be provided as a pair, or the implementation is undefined). If provided, the default behaviour [where the text selector might be shown and hidden automatically] is not performed. Instead the given callback methods will be invoked at the start and end of this participant's (contiguous) speech. Both callback functions take the form:

```
talkFn( conversation, stage)
```

and *'this'* will contain the participant object. If styling is not required it may be set to *null* or *{}*.

For example, to show the participant's speech selector (a behaviour similar to the overridden behaviour), *talkStartFn* might be:

```
function talkStartFn(conversation, stage) {
    document.querySelector(this.target).style.display = 'block';
}
```

or in jQuery:

```
function talkStartFn(conversation, stage) {
    $(this.target).show();
}
```

A similar *talkEndFn* could provide the hiding functionality.

addStage();

This adds a single stage of conversation for a participant. This will often be a single line of text. Multiple calls to *addStage()* during construction define the conversation. The first argument to *addStage()* is normally the participant.

There are several forms which this method can take:

addStage(participant, textOrHTML);

participant — is the participant who say this line of text. It must be an object returned from a call to *defineParticipant()* prior. Details: it is a participant object.

textOrHTML — This is the text (or HTML) which this participant says at this stage of conversation (generally a single stage is a single line of text, presented at once). If HTML tags are found within the string it will be interpreted as HTML rather than plain text (though you can override this).

Example:

```
let sara = conversation.defineParticipant(...); // prerequisite
conversation.addStage(sara, "Hello there");
```

When this stage is reached (in active conversation) the provided text or HTML will be inserted into the participant's text selector.

addStage(participant, data, callback);

This variation does not insert text into the participant's selector (as in the first variation). Instead when the stage is reached a callback function will be invoked, allowing the user to provide custom behaviour. By way of example, to perform behaviour similar to the custom behaviour (where text/html is inserted into a DOM element when this participant is talking) the callback method might be:

```
function stageCallback(conversation, participant) {
    document.querySelector(participant.target).innerHTML = this.data;
}
```

or in jQuery:

```
function stageCallback(conversation, participant) {
```

```

    $(participant.target).html(this.data);
}

```

You can provide '*data*' as *null*, or use it for your own purposes (in which case it can be any type, not just a string). You can access this data in the callback function as `this.data`.

`addStage( { properties } );`

This variation adds all stage properties as a list (as key/value pairs). This reveals extra configuration. The keys for properties are:

label	An optional string used to identify this stage, allowing us to jump directly to this part of the conversation (using <code>gotoLabel()</code>).
participant	The conversation participant who says this line of text (see <i>data</i>). It is the value returned from a call to <code>defineParticipant()</code> .
data	This is what the participant says, at this stage of conversation. It is typically a string of text or HTML (automatically determined). This automatic determination can be overridden using <i>isHTML</i> . Should the user provide a callback function for key ' <i>start</i> ', then the data may be any item (not just a string) and used for custom purpose, accessible to the callback function.
isHTML	A boolean value to override the text or HTML determination for property ' <i>data</i> '.
start	This is a callback function. If provided, the text/HTML placement described above will not be performed. Instead this function will be called when the stage is reached. The function signature is:

```
function dataFn(conversation, participant)
```

Any data provided may be accessed within this function as `this.data`.

Detail: any other properties you provide will be retained, and can also be accessed via `this`.

When a stage is shown, by default if the text selector or any ancestor is hidden, it will be shown for the duration of displaying the speech, and restored to hidden afterwards.

`label();`

A label is simply a "go to" point, which you move conversation to using the `gotoLabel()` method.

`label( name );`

This is a convenience method. The single argument is a string, and will be applied to the next stage defined (next call to `addStage()`). So the next stage will take on this label.

This method exists purely for neat conversation code formatting, as it may be interspersed with calls to `addStage()` without the need to use the long-form (less readable) `addStage()` method. Eg.

```

addStage(...)
addStage(...)
label('fred yells')
addStage(fred, "<strong>Get out</strong>")

```

```
addStage (...)  
addStage (...)  
...
```

start();

Start the conversation. Conversation will advance to the first stage defined. For simple conversations this method is optional, as a first call to next() will have the same effect. This will not restart a conversation, even if it has been completed.

start();

This method takes no arguments.

next();

Go to the next stage in conversation. When there are no more stages it will end then conversation.

next();

This method takes no arguments.

gotoLabel();

Go to a predefined label (a label belongs to a conversation stage). See addStage() and label() methods.

gotoLabel(labelName);

This method takes a single string parameter, being the label name to move to. Conversation will continue sequentially from this point.

reset();

Stop conversation progress and return to the beginning.

reset();

Reset gracefully, see below.

reset(graceful);

Reset gracefully or not. ('graceful' is a boolean). A graceful reset will call 'end' user functions as should typically be expected to end the conversation. The conversation can now be restarted if required.

Ungracefully simply aborts the conversation and resets conversation to the start. Callbacks are not invoked. The state of the page may or may not permit a graceful conversation restart.

end;

This property (not a method) may be set on the conversation object to provided a callback method, to be invoked at the end of conversation. The method takes no parameters. It will also be invoked from a graceful reset (see `reset()`, documented above).

```
conv.end = function() {  
    // perform clean up  
}
```

setStyles();

This method is in the global scope (not a method of the conversation object. It allows you to easily apply multiple styles to a DOM element in Javascript, similar to the functionality provided by jQuery's `.css()` method.

`setStyles( selector, styleList );`

The style list is a object containing CSS key/value pairs. It will be applied to all DOM elements specified by the selector.

Final note:

Further functionality can be obtained by reading the *conversation.js* code, however any undocumented behaviour might not remain available for future versions.