ResponsIF

Riff Phrase Book

Version 0.65

Table of Contents

Revision History

Version	Who	Date	Change
0.6	J. Nabonne	2016-06-06	Initial revision.
0.65	J. Nabonne	2016-06-23	Added “onto”. Revised “as” description to match code change.

Introduction

This document provides a reference for all the current riff phrases – those phrases which go into the creation of a riff. A riff is a sequence of dotted tokens with associated values. Values for some tokens are optional. A riff phrase is a token/value pair, and looks like the following:

```
.token the value for the token
```

In this example, ".token" is the token, and "the value for the token" is its corresponding value. Riff phrases are combined in various ways to create a riff.

In the following example (not real riff tokens), there are three phrases: ".finds Henry", ".doing work", and ".on Sunday".

```
.finds Henry .doing work .on Sunday
```

Content in a riff is encoded exclusively as phrases, with each begun with a dotted token. (So, for example, you could not have "Sarah .finds Henry .doing work" because "Sarah" is not a dotted token.)

Meta Phrases

This section details phrases which may exist anywhere in a riff and operate outside the parser. They are handled by the riff pre-processor, during the riff "expansion" phase before parsing.

Comments

Any phrase beginning with the following characters is considered a comment and stripped from the stream before parsing: '/', '-', or '#'. Any characters may follow the initial comment character, and the value of the comment is stripped along with the phrase. Note that adding one of those characters to the beginning of the phrase will only remove that phrase. There is no current way in a riff to "comment out" a block of phrases without touching each phrase.

VALUE: any desired text

EXAMPLE:

```
.------------------
.- This is a comment
.------------------

.////////////////////
.// This is a comment
.////////////////////

.####################
.# This is a comment
.####################

.# The following phrases will be ignored as comments.
.-response responder
.-end

.---- As will these
./actions
./end
```

.include <file>

Description: Includes the contents of the specified file at that location.

Value: the file to include

Example:

```
.include kitchen_responses.txt
```

.define <macro>

Description: Begins the definition of a macro phrase. All phrases up to a closing ".enddef" phrase are added to the definition. The phrase ".<value>" will be replaced when the phrase is used later with the value of the invoking phrase. (For example, if the macro is named "remember" and it is invoked with ".remember me", then any instance of .<value> in the definition will be replaced with "me".)

While macro definitions can technically occur anywhere in the riff stream, it might make more conceptual sense to keep them at the top level in a file (not buried deep within a response, for example).

Value: the name of the new macro phrase.

Contains: any desired phrases.

Example:

```
.define moves-north-to
    .moves .to .<value>
    .sets last_direction .to north
.enddef

.// Later, use as
.moves-north-to kitchen
```

Top-level Phrases

This section details phrases which may exist at the top level of a riff.

.setup <responder>

Description: Defines the actions to be performed to set a responder's initial state. Note that there is no "end" phrase for "setup". The block is automatically closed when any non-action phrase is encountered. The optional responder value specifies the context for the actions being executed. There may be more than one setup block and more than one with the same responder. Setup blocks are processed in the order they're encountered in the riff.

Value: the responder to reference (optional)

Contains: any Response Action phrases

Example:

```
.setup janitor
    .sets name .to Janitor    .// sets janitor:name="Janitor"
    .moves .to cafeteria      .// janitor is used as responder
```

.responses <responder>

Description: Defines responses for a responder. This phrase begins a response group, which must be closed with an ".end" phrase.

Value: the responder

Contains: any Response Group phrases

Example:

```
.responses Darth Vader
    .response father
        .does .says "No... I am your father."
.end
```

.actions <responder>

Description: Defines actions for a responder. This phrase begins a response group, which must be closed with an ".end" phrase.

Actions are processed after each riff turn.

Value: the responder

Contains: any Response Group phrases

Example:

```
.actions dog
    .response
        .does .says A dog barks in the distance.
.end
```

.click-effect

Description: Defines the default animation effect applied when a link is clicked.

Value: not applicable

Contains: Animation phrases

Example:

```
.click-effect .to {"color": "#c0c000"} .lasting 250 .to {"color": "#fff"} .lasting 300
```

.model <responder>

Description: Defines the model for a responder. This phrase begins model definition, which must be closed with an ".end" phrase.

Value: the responder

Contains: any Model phrases

Example:

```
.model dog
    .cluster learned-behavior
```

```
    .cluster instincts
```
.end

Definition Phrases

The following phrase has special meaning within a macro phrase definition. Normally, any phrases within a macro definition are expanded verbatim when the macro is used. However, this phrase allows a small amount of parametrization through the insertion of a value in one or more places in the expanded macro.

.<value>

Description: Inject the value of the macro invocation, attaching to the previous phrase's value.

Value: additional value to be appended after the macro value is set or appended (optional).

Example:

```
.define says-text
    .does .says .<value>
.enddef
```

If this is invoked with `.says-text This is some text`, then the resulting expansion will be:

```
.does .says This is some text
```

Similarly, given the following macro

```
.define says-text-again
    .does .says .<value> again.
.enddef
```

if it is invoked with `.says-text-again This is some text`, then the resulting expansion will be:

```
.does .says This is some text again.
```

Response Group Phrases

These phrases can occur within a response group.

.response <topics>

Description: Begins the definition of a response.

Value: the topics to match (optional)

Contains: any Response phrases

Example:

```
.response sunset
    .needs love
    .does .says You cuddle up on the beach as the sun bids good night.
```

.reference <responder>

Description: Inserts a reference to another responder's responses. When responses are being processed,

the other responses will be included. Note that the actual responses are used, not copies, so things like the occurrence count will be shared. Also, the original responder will be used, not the responder for the referenced responses. (So if responderA references responderB's responses, responderA will still be considered the current responder even while processing responderB's responses.)

VALUE: the responder whose responses are to be included.

EXAMPLE:

```
.responses king
    .reference male
    .reference human
.end
```

.end

DESCRIPTION: Signals the end of a response group.

VALUE: not applicable

EXAMPLE:

```
.responses car
    .response start
        .does .says The car turns over slowly and then roars to life.
.end
```

Response Phrases

These phrases can occur when defining a response.

.matches <topics>

DESCRIPTION: Provides an alternate (older) way of specifying the topics for the response to match, rather than as the value of the response.

VALUE: the topics to match

EXAMPLE - the following two responses are equivalent:

```
.response topicA topicB
    .does .says Something

.response
    .matches topicA topicB
    .does .says Something
```

.occurs <count>

DESCRIPTION: Specifies the maximum number of times a response will run. Once the response's run count matches the “occurs” count, the response will no longer be considered eligible to run.

VALUE: the maximum number of times the response will run before becoming ineligible.

EXAMPLE:

```
.response door
    .occurs 1
```

```
    .does .says The door is wooden and seems solidly built.
```

.needs <state>

Description: Specifies the needs for the response. This is an expression that is evaluated. If the result is greater then 0.5 for a numeric value, or non-empty for a non-numeric value, then the needs are satisfied. A response's needs must be satisfied for the response to be considered eligible to run.

Value: an expression to evaluate to deter

Example:

```
.response panic
    .needs afraid and not brave
    .does .says The mouse scurries under a nearby table.
```

.prompts <prompt>

Description: Assigns a prompt to a response. When the response is executed, the prompt will be shown first to the player. If the prompt is selected, then the response's actions will execute. This allows a level of confirmation before a response does what it does.

Prompts can have mark up.

Value: the prompt to show to the player

Example:

```
.response door
    .prompts Look at the door
    .does .says The door is wooden and seems solidly built.
```

.forcesprompt <value>

Description: There may be times when it's desirable for a response to only show its prompt if the player is being given a choice (that is, there is more than one choice to be made). Otherwise, the response should just execute. By default, a response with a prompt will always show the prompt, even if it's the only one. By setting “forces prompt” to false, the response will simply execute if it's the only prompt rather than showing its prompt. If there is more than one choice, the prompts will be shown as usual.

Value: true or false (optional. Defaults to true.)

Example:

```
.response door
    .forcesprompt false
    .prompts Look at the door
    .does .says The door is wooden and seems solidly built.
```

.is <class>

Description: Assigns a class to a response. Responses are grouped by classes when being processed.

Value: the class to assign to the response

Example:

```
.response hint
    .needs saw-stairwell
    .is hint
    .does .says Have you tried going upstairs?
```

.selects

Description: Begins a “selects” block. When responses are being gathered for processing, responses within the “selects” block are included, if the parent response is eligible. This allows responses to be grouped by topics or by needs.

“Selects” is applied during the response *selection* phase, while responses are being scored. For a way to group responses at response *execution* time, see “uses”.

A “selects” block is a response group and must be ended with a “.ends” phrase.

Value: not applicable (may change at some point to have values like “all”, “first”, etc)

Contains: any Response Group phrases

Example:

```
.response *north
    .needs door_open
    .selects
        .response .does .says A response to consider.
        .response .does .says Another response to consider.
    .end
```

.orders <index>

Description: The orders phrase allows response execution to be ordered. It is usually used to control the output of order where output is coming from different responders, but it can be used to control execution order in general.

The default order value if not specified is 1. Greater values will be processed after those with lower values. Responses with the same value will be processed in the order encountered.

Value: a number.

Example:

```
.response test
    .orders 2
    .does .says This text will appear after normal response "says".
```

.weights <expression>

Description: Specifies a weight to be applied to the calculated topic score. The weight should be a numeric expression (constant, equation, etc). The topic score is multiplied by the result when scores are being calculated. (Note that a response without topics has a default score of 1.)

A response without a specified weight has a default weight of 1.

Value: an expression

Example:

```
.response panic
```

```
    .weights 0.5
    .does .says The mouse scurries under a nearby table.

.response panic
    .weights bravery
    .does .says The mouse rears up and looks you in the eye.
```

.does <occurrence>

Description: Begins the definition of a response "does" block. A "does" block contains the actions that a response actually does when executed.

There can be multiple "does" blocks, each with a different occurrence value. The block matching the current occurrence will be executed, if there is one. If there is no block matching the current occurrence, then a block without an occurrence will be used, if there is one. If neither a block with a matching occurrence or default block exists, the response does nothing.

Value: the occurrence number for the block (optional)

Contains: any Response Action phrases

Example:

```
.response sunset
    .does 1
        .says You gaze upon the setting sun and then continue up the beach.
        .moves .to north beach
    .does
        .says The sun is nearly set for the day. .autohides
```

Response Action Phrases

.says <text>

Description: Outputs the specified text.

Value: the text to be output

Example:

```
.response walls
    .does
        .says High stone walls reach to the {!ceiling!]. It's hard to
            believe these are man-made, as they reek with the stench of {!magic!}.
```

Optional Phrases: the following phrases can be optionally used with “says”, in any combination.

.into <selector>

Description: Specifies an HTML element to “say” the text into. The selector is a standard CSS selector, though the usage of CSS class names is problematic due to a leading '.' being interpreted as beginning a new riff phrase.

Any content in the element is replaced with the next text.

Value: the CSS selector for the element

EXAMPLE:

```
.response update_inventory
    .does
        .says {+INVENTORY+} .into #inventory-pane
```

.onto <selector>

DESCRIPTION: Specifies an HTML element to "say" the text onto the end of. The selector is a standard CSS selector, though the usage of CSS class names is problematic due to a leading '.' being interpreted as beginning a new riff phrase.

The specified text is appended to any existing text in the element.

VALUE: the CSS selector for the element

EXAMPLE:

```
.response next
    .does
        .says some more .onto #container
```

.as <style>

DESCRIPTION: Specifies the style for the output text. The style must be defined as a CSS class. This phrase works even when responses are invoked via call markup: the text has the correct style applied.

VALUE: the CSS class to be applied to the text

EXAMPLE:

```
.response
    .occurs 1
    .does
        .says Click on any highlighted text to explore. .as help-text
```

.autohides

DESCRIPTION: Marks the output text as "auto-hiding",: the text will disappear on a subsequent turn. Auto-hide text is useful for text that is needed for the current response but is not desired in the long-term story transcript.

VALUE: not applicable

EXAMPLE:

```
.response monster
    .does 1
        .says The monster is large and hairy, with long, sharp teeth.
    .does
        .says The monster continues to scare you. .autohides
```

.sets <state expression>

DESCRIPTION: Assigns a value to a state variable. The value is a "set expression". There are a number of

different expression types possible. They are detailed below.

VALUE: “set expression”

.sets <variable>=<value expression>

DESCRIPTION: Assigns a value to a state variable. The value expression can be a simple value or an arbitrary expression, as long as it evaluates to a value.

EXAMPLE:

```
.response
    .does
        .sets age=31
        .sets seven=3 + 4                ./ seven now equals 7
        .sets first name="Billy"
        .sets last name="Ball"
        .sets full name=first name + " " + last_name
        .sets happy=satisfied and content
        .sets death=age+45
```

.sets <variable> .to <string>

DESCRIPTION: Assigns a string value to a state variable. The “.to” value is assigned directly as a string, with no interpretation. This allows the setting of string literal values without needing to surround thm with quotes (which can be convenient if the string itself contains quotes). The statement

```
.sets variable .to value
```

is equivalent to

```
.sets variable="value"
```

EXAMPLES:

```
.response
    .does
        .sets age .to 31                 ./ age now equals the string "31"
        .sets seven .to 3+4              ./ seven now equals the string "3+4"
        .sets first name .to Billy       ./ name now equals "Billy"
        .sets death=age+45               ./ now equals the string "age+45"
```

.sets <variable>

DESCRIPTION: Sets the designated variable to true (1).

EXAMPLE:

```
.response
    .does
        .sets happy                 ./ happy now equals 1
```

.sets not <variable>

DESCRIPTION: Sets the designated variable to false (0).

EXAMPLE:

```
.response
```

```
    .does
        .sets not alive             ./ alive now equals 0
```

.sets un <variable>

Description: Sets the designated variable to “un” (-1).

Example:

```
.response
    .does
        .sets un grateful           ./ grateful now equals -1
```

.sets more <variable>

Description: Increases the variable toward 1.

Example:

```
.response
    .does
        .sets more hungry
```

.sets less <variable>

Description: Decreases the variable toward -1.

Example:

```
.response
    .does
        .sets less intelligent
```

.calls <topics>

Description: Calls the specified topics.

Value: a “weighted topic list” of topics to call

Example:

```
.response ROSE_NORTH
    .prompts Move to the kitchen
    .does
        .says You head north to the kitchen.
        .calls LOOK
```

.moves <responder> .to <responder>

Description: Assigns a responder a new parent. In order for this to be meaningful, an additional “.to” phrase should be present specifying the new parent.

Value: the responder to “move” (optional). If not specified, the current responder is used.

Example:

```
.response jail
    .does
        .says Go directly to jail. Do not pass Go.
        .moves .to Jail
```

```
.response add hotel
    .does
        .says You add a hotel to Boardwalk.
        .moves hotel .to Boardwalk
```

.suggests <topics>

Description: Suggests the specified topics to all suggestible responders in the current response context.

Value: a “weighted topic list” of topics to suggest

Example:

```
.response kiss
    .does
        .says Unexpectedly, Skye moves close and gives you a tender kiss.
        .suggests love=0.9 passion=0.6 happiness=0.8 confusion=1
```

.adds <topics>

Description: Adds the specified topics to the cluster for a responder. If no additional phrases are used, the topics will be added to the default cluster for the current responder. A “to” phrase can be used to specify which cluster to target, and a “for” phrase can be used to specify the responder.

Value: a “weighted topic list” of topics to add

Example:

```
.response learn_karate
    .does
        .adds karate
        .adds judo .to goals
        .adds satisfaction .to emotions .for teacher
```

.removes <topics>

Description: Removes the specified topics from the cluster for a responder. If no additional phrases are used, the topics will be removed from the default cluster for the current responder. A “from” phrase can be used to specify which cluster to target, and a “for” phrase can be used to specify the responder.

Value: a “weighted topic list” of topics to remove (though the weights don't matter)

Example:

```
.response forget_karate
    .does
        .removes karate
        .removes martial_arts .from goals
        .removes satisfaction .from emotions .for teacher
```

.animates <selector>

Description: Animates an HTML element. The element is specified by a CSS selector. This phrase should be followed by one or more “to” phrases to specify the animation effect to apply, as well as any “lasting” phrases to specify time durations. Durations are specified in milliseconds (1000 equals 1 second, 500 equals half a second, etc).

Animation effects are jQuery animations, specified in JSON format. Each “to” specifies a single effect, listing attributes and target values. The “lasting” phrase will dictate the duration of the transition to the target values. (A “to” without a “lasting” duration will take effect immediately.)

For further examples of possible animations, see any of the various jQuery animation websites.

VALUE: the CSS selector of the element to animate

EXAMPLE:

```
.response show
    .does
        .animates #title
        .to {"font-size": "0%", "opacity": "0"}
        .to {"font-size": "100%"} .lasting 1000
        .to {"opacity": "1"} .lasting 1000
```

.invokes <JavaScript>

DESCRIPTION: Invokes JavaScript code. For more details, see the ResponsIF User's Guide.

VALUE: the JavaScript code to invoke.

EXAMPLE:

```
.response jail
    .does
        .invokes
            console.log('Invoking some JavaScript code');
            alert("I've responded!");
```

.resets

DESCRIPTION: Sets the current response's occurrence count back to 0. This allows (among other things) responses to be cyclic.

VALUE: not applicable

EXAMPLE:

```
.response poke
    .does 1
        .says 1.
    .does 2
        .says 2.
    .does 3
        .says 3. And back to the beginning...
        .resets
```

.adjusts <variable> .toward <expression> .stepping <expression>

DESCRIPTION: Adjusts a variable toward a target value. The step size can be specified as well. For more information on the mathematics behind this, see the ResponsIF User's Guide.

VALUE: the variable to adjust

EXAMPLE:

```
.response
    .does
```

```
        .adjusts excitement         ./ unspecified .towards is 0. Step is 0.2
        .adjusts happiness .toward 0.5
        .adjusts anger .toward irritation or fear .stepping 0.1
```

.clears

DESCRIPTION: Clears all content from the default output element.

VALUE: not applicable

EXAMPLE:

```
.response continue
    .prompts Go to chapter 2
    .does
        .clears
        .moves .to chapter2
```

.uses

DESCRIPTION: Begins the definition of responses to be used as actions for the containing response. This phrase begins a response group, and it must be ended with a ".ends" phrase. How the responses are used depends on the type.

VALUE: defines the type of usage. The various types are defined next.

.uses all

DESCRIPTION: All the eligible responses in the "uses" group will be processed, in the order they appear in the group.

EXAMPLE:

```
.response
    .does
        .uses all
            .response
                .does
                    .says This response will always be processed.

            .response
                .needs state
                .does .says This response will be processed if "state" is set.

            .response
                .does .says And this response will always be processed.
        .end
```

.uses first

DESCRIPTION: The first eligible response in the group will be processed. Any responses after the first eligible response will not be examined.

EXAMPLE:

```
.response
    .does
        .uses first
```

```
            .response
                .needs state
                .does
                    .says This response will be processed if "state" is set.

            .response
                .does
                    .says This response will be processed if "state" is not set.
        .end
```

.uses random

DESCRIPTION: A response will be selected randomly from the eligible phrases in the group.

EXAMPLE:

```
.response pen_color
    .does
        .uses random
            .response .does .sets pen .to blue
            .response .does .sets pen .to black
            .response .does .sets pen .to red
            .response .does .sets pen .to green
        .end
```

.uses priority

DESCRIPTION: A response will be selected using the same criteria used to select responses normally: all the eligible responses are scored, and the responses with the highest score will be executed. The responses executed are considered the "priority" responses. (Contrast this with ".uses best" which only executes a single priority response.)

EXAMPLE:

```
.response what_next
    .does
        .uses priority
            .response hungry or tired or aroused
                .does .adds looks-for-a-hotel .to short_term_goals
            .response hungry and irritated
                .does .adds looks-for-food .to short_term_goals
            .response irritated or stressed or depressed
                .does .adds looks-for-bar .to short_term_goals
            .response bored or favorite_sport_on_tonight
                .does .adds looks-for-television .to short_term_goals
        .end
```

.uses best

DESCRIPTION: This is similar to ".uses priority" except that only a single response will be executed. If there is only a single highest scoring response, that response will be executed. Otherwise, a single random response will be chosen from the priority response.

EXAMPLE:

```
.response
    .does
        .uses best
            .response happy .does .says Uncle Andrew smiles.
```

```
        .response sad .does .says Uncle Andrew looks at you sadly.
        .response angry .does .says Uncle Andrew can barely contain his rage.
        .response sleepy .does .says Uncle Andrew yawns and stretches.
    .end
```

Model Phrases

The following phrases are used in the definition of responder models. A responder model defines how a responder gathers, and modifies over time, topics.

.cluster <name>

Description: Defines a model "cluster". A cluster is a group of topics, grouped for some particular reason. Examples of clusters are "short term topics", "long term topics", "emotions", "goals", "instincts", "motivations", etc.

A cluster has a name. The topics from all clusters within a responder model are combined for each call.

A cluster can have an overall weight. It can also have a decay factor, which causes the topic weights to adjust toward zero on each game cycle (for transient, short-term topics, for example). Finally, clusters can be labeled as "suggestible", which means they will be the cluster where any suggested topics go.

Example:

```
.model Sebastian
    .cluster long_term .weight 0.9
    .cluster short_term .decay 0.1 .suggestible
.end
```