Validation Prototype Notes

Author: Henrik Mikael Kristensen

Revision History

22-Jul-2010	A

Table of Contents

1. Notes

The aim of the validation prototype is to provide a complete and almost transparent validation system for the R3 GUI. It follows most of the specs (see the Validation Proposal in this directory).

Note that we may not achieve everything we want in the first prototype. This will be determined after a review.

The file `validation.r3` contains the prototype and has a test function at the end, `gui-test`, which will open a simple window with some faces in it that can be validated and indicated.

1.1. Facts

- Around 300 lines of code without comments and a few bits of dead code.
- It currently is not wrapped in a context as I’m not sure this is what is needed to do.
- The dialogs system prototype will require this as some dialogs use form validation.
- It requires the tag system from `style-tags.r3`.
- The design is largely based on how reactors are built and how the database record prototype works to keep the usage familiar.
- The UI is currently very ugly and not really usable. This requires some style changes by the new resizing scheme for it to become pretty again.
- Validation is a 2-step process: First validate the face(s), then display the validation result using indicators or reports.

1.2. Usage

The validation details in a layout is only used at validation time, i.e. details are gathered very late. This therefore happens every time you want to validate the form. Late gathering is done in case validation details change during the life time of the layout.

Validation Options

Validation details are specified as options in the layout. The options must be applied to the faces which must be validated or to panels, which contain faces to be validated.

There are 3 standard options available:

skip	This is identical to the database field option `skip` on purpose and both validation and the database record collection function will use them. When `skip` is used, the validation is skipped for this field or the entire panel. This overrides any other validation information in the options for the face.
validate	This is a word or a block of words. Anything else is ignored. The input here are **validators** (see below). If using a block, the validators are evaluated in sequence and validation stops on the first validation error for that face. When encountering an unknown validator, an error is thrown.
required	This tells whether the validation is required to pass for the face. If it does not pass for a required face, the validation for the panel is considered invalid, allowing you to stop the user from continuing a form submission. The input is `logic!`.

Furthermore: Validation only occurs for faces that have a name and have tags `info`, `field` and `state`. This is determined using the internal `validate-face?` function. These requirements are not necessary for panels, as the validation details are gathered anyway.

Even though you are free to pass validation options to a panel, it may not mean that the panel itself is validated as its value description is an object. It still needs the above tags, which a standard `GROUP` or `PANEL` style does not provide.

Validation Results

Each face that is validated, stores its own validation results in `facets/validate-result`. If the face is not up for validation, i.e. does not have a name or the above mentioned tags, the face won't have any results stored. Results are also not stored if the `skip` option is used and in case of that, `facets/validate-result` may not exist and will return NONE if used attempted fetched with `get-facet`.

The result is always one of four words:

not-required	This is for fields which are not required and is set during initial validation and will also show for faces that do not pass validation.
required	This is for fields which are required, but are unfilled and unedited. If `validation/init` returns this result here, the form is invalid for submission. The function `valid-panel?` will return `FALSE` on first encounter of this result.
valid	The field is valid for submission. This goes both for required and not-required fields.

invalid	The field is invalid for submission. This goes for required fields that fail validation and is considered a fatal error, and `valid-panel?` will return `FALSE` on first encounter of this result.

All results are also stored in plain English in `facets/validate-error`. The strings for these are determined by validators and can (when tool-tips are implemented) be read as a tool-tip in the indicator in the layout or read via the report dialog, when that is implemented.

Validation Methods

You are free to add new validation methods or **validators**, which will work globally for the R3 GUI. The procedure is similar as for creating reactors, in that you provide a word with a code block, which then safely is added to the global list of reactors usable in layouts.

The prototype offers several standard validators:

only-chars	The input may only contain characters, not numbers. The comparison happens against a bitset.
not-empty	The input is assumed to be a series and may not be an empty series.
only-numbers	The input may only contain numbers. The comparison happens against a bitset.
only-integers	The input may only contain integers 0-9. The comparison happens against a bitset.
only-positive	The input may only contain positive numbers. The input is assumed to be a number, so use this after using `only-numbers` or `only-integers`.
selected	The input may not be NONE or an empty `any-block!`, which could be a list, where at least one item is required for selection.
email	The input must be parsable as an email address.

The `make-face-validator` function is named after the `make-face-action` function. It accepts three arguments:

word	The name of the validator.
string	The description of the validator. This is optional.
block	The code for the validator, not as a function but as a block. The reason for using a block without an function header is that the argument list is fixed as `face` and `value` and you don't provide arguments in the validators block. The outcome of the code block is internally converted to `logic!`, when the code block is evaluated.
string	This is an error string. Unfortunately the way to write this string is a bit contrieved, but the intent is to have it used along with the name of the field or panel we are validating. Thus the form of writing of this string is a partial message: <tt>"is empty"</tt>. When written like this and the face name is <tt>Name</tt>, the output will be <tt>"*Name* is empty"</tt> when passed through the <tt>validate-error-string</tt> function.

Example:

```
make-face-validator [
        not-empty
        "The field must contain a series and it may not be empty."
```

```
        [all [series? value not empty? value]]
        "is empty"
]
```

Any existing validator will be overwritten.

Scoping

Validation details are scoped, which means if you state specific details for a panel, everything inside it, will use those details unless a change is made inside that panel. Scoping is entirely inward, not downward or outward.

Validation details are placed in a stack inside the validation function. When the panel or face you have decided validation details for has been through validation, the details are removed from the stack, revealing the previous level of validation details.

As a side note, the database record system, designed as a different prototype, available in `db-reactors.r3`, works in exactly the same way. It uses the same scoping method.

Examples

All fields in this panel may not be empty and all fields are required.

```
panel [
        name: field
        address: field
        city: field
] options [validate: [not-empty] required: true]
```

Here, the `city` field is not required:

```
panel [
        name: field
        address: field
        city: field options [required: false]
] options [validate: [not-empty] required: true]
```

Here, the `age` field must be a positive integer. Note the order of the validators for that field. If the order is reversed, the program will crash when `value` is not a number, because the `only-positive` validator requires the input to be a number:

```
panel [
        name: field
        address: field
        city: field options [required: false]
        age: field options [validate: [only-integer only-positive]]
] options [validate: [not-empty] required: true]
```

Note that validators are replaced, when scoping inward. They are not appended to the previously stated validators.

Scoping should also work for panels inside panels:

```
a: panel [
        b: panel [
                field
```

```
                field
        ] options [required: false]
        c: panel [
                field
                field
        ] options [validate: [only-integers]]
        d: panel [
                field
                field
        ]
] options [validate: [not-empty] required: true]
```

- Panel `a` holds that all fields inside it are required and may not be empty.
- Panel `b` holds that the previously stated required is revoked for fields inside it.
- Panel `c` holds that validation is changed to be only for integers for fields inside it. All fields inside it are required.
- Panel `d` holds that it uses the same validation details as panel `a`.

1.3. Validation

Validation usually comes before indication.

Validating

There are three validation functions:

validate-panel

This function validates a panel and is generally used when a form is first opened and when an attempt as submission is made. It traverses the face and each `FACE/FACES` block recursively and submits each face for validation. When the face is validated, the result is immediately stored in the face.

The `/init` refinement validates according to initial condition. This means that fields that are required and filled, will be marked `valid`, while fields that are required and unfilled, will be marked `required`.

This refinement is used when displaying the form for the first time and it's useful after just creating the layout or when using `reset-face` and `set-face` on an existing form. Therefore, it can both be used when the form is clean and when the form is pre-filled with content.

validate-face

This function validates a single face and is generally used when a field needs to be validated, when you are editing it or are unfocusing it.

The function works by gathering the options from its parent faces continually until all required options or the window face is encountered and then performing the validation as normal.

This can be a little bit heavy, since this is done on each call, so `validate-face` should be used sparingly for now.

valid-panel?

This function returns `logic!` for whether the face or panel is considered valid for submission. Any fatal error, such as encountering an `invalid` or `required` face will result in returning `FALSE`. This function is used during submission of a form.

Reactors

Then there is the `validate` reactor, which is used, when validating a single field based on when it's unfocused or when actions are run on it. We are not yet entirely in control over when reactors are run, currently not possible to do per keystroke.

Reactors are not scoped, i.e. they must be stated for each face which you want to validate by unfocusing it. It won't work on panels.

Example:

```
view [
        label "Name"
        field validate options [validate: 'not-empty]
        indicator
]
```

Internal Functions

set-validate-result	This sets the validation result for a face.
validate	This determines the required validators for a face and evaluates each validator.
validate-error-string	This provides a human readable string for the error or status of the face validation result.
validate-face?	A function to determine if the given face is eligible for validation.

1.4. Indication

Indication usually comes after validating.

Indicating

When indicating the result of a validation of a single face, use the INDICATOR style. This style will eventually present one of 4 possible images:

VALID	(green checkmark)
INVALID	(red X)
REQUIRED	(filled dot)
NOT-REQUIRED	(hollow dot)

The indicator is currently just displaying a +FORM+ed word for the status, not an image.

It's not going to be standard to have a face indicate validation inside itself. I considered that this would require that all styles need to be redesigned for this feature, and that would be both a

considerable amount of work and also would make it harder for new users to design styles that adhere to validation.

However it would be possible to provide an `ON-VALIDATE` actor and have that actor respond to `FACETS/VALIDATE-RESULT`, if you really want validation to be done inside one single style.

Indicate Panel

Updating the indication occurs with `indicate-panel` and is performed separately from `validate-panel` and `validate-face` to allow control over when you want the validation result to be visually updated.

The use would be first to perform a `validate-panel` or `validate-face` and then perform `indicate-panel`.

The function is used inside `create-dialog` in the dialog prototype for now, but would be used inside `view` later. Currently `view` is too monolithical to provide this on its own, and a workaround is used inside the `create-dialog` function in `dialog.r3`. This will change later.

The function also focuses on the first face that fails validation or requires validation, in sequentially less fatal order:

1. Find the first `INVALID`
2. Find the first `REQUIRED`
3. Find the first `NOT-REQUIRED`
4. Find the first `VALID`
5. Find the first face which is not eligeble for validation but can be filled anyway. This is not yet implemented, as the policy for this is not yet fixed inside `view`.

Indicate Face

When performing validation of a single face, it’s only necessary to indicate for a single face, so the `indicate-face` function is used.

This function is used inside `indicate-panel` and in the `validate` reactor.

Facts

- The use of the indicator style in your layout is voluntary, but it’s necessary, if you want the user to see which field fails validation, unless you want to use reports.
- Each indicator will be equipped with a tool-tip to explain the current status of the validation evaluation, once the help system is implemented, but at this time, the status is already assigned in `FACETS/TOOL-TIP`.
- The indicator must be placed right after the field you wish to validate and in the same `FACES`-block. I’m not sure there is a better way to deal with that. For best appearance, the indicator should be placed visually to the right of the face you wish to validate.

Reporting

There may be cases where you want a report of validation details for a dialog, which states which fields failed and why in a single view. I've found it necessary to include such a feature on some occations where the form is very complex or has an apparently illogical flow and indicators are not enough.

Using the `report-panel` function, all fields that have the status `invalid` or `required` will be listed in this report. The report is returned as a block of the form:

```
[
        face-name [word!]
        validate-result [word!]
        human-readable-result [string!]
]
```

Then you are free to perform your own reporting scheme, if not using the planned dialog with `request-report` (not yet implemented).

The function will try to form the human-readable error string by looking at the face that comes *before* the validated face, which should be a label face. If one is found, its text is used as the name of the face in the error string. If one is not found, the name of the validated face is used.

When the form is OK, `report-panel` will return an empty block. It will also return an empty block if the panel is empty or contains no faces that can be validated.

2. Problems

The prototype has a few problems:

- Validation of any type of field by user input, i.e. per keypress or per unfocusing using validate-face is not yet clear, how that should be done.
- Evaluation is largely lazy to make sure that all validation information for a layout is current and correct, so there are times when a lot of work is done, such as inside `validate-face`.
- The human readable result relies on the name of the face, i.e. the set-word! used in the layout. Thus it may be difficult to localize or it may be difficult to deciphre if the word uses an obscure naming scheme, i.e. `f-db-city`. A method needs to be found to use a human name and I've been thinking about using the previously occuring label text, but it is not a solid method.
- Forms generally require the arrangement of three faces **label**, **field**, **indicator**, **label**, **field**, **indicator**, … etc. setup for validation to work 100%. It's possible to change this, but I'm afraid it will mean that many more options will need to be filled for the layout.
- Unable to really test whether fields can do inline validation, since the tab navigation is broken. This has been tested using buttons triggering the `validate` reactor.
- `indicate-panel` is not capable of focusing the first face through standard window opening requirements as those have not yet been described elsewhere in the system.