Validation Proposal

Author: Henrik Mikael Kristensen

Revision History
25-May-2010 A

Table of Contents

1. Rationale 1
2. Ideas 1
    2.1. VALIDATE-FACE 1
    2.2. Validation indication 1
    2.3. Validation states 2
    2.4. Validation modes 2
    2.5. When to validate 2
3. Source examples 2
4. Notes 3

1. Rationale

The R3-GUI needs a way to validate forms. Form validation is trivial and time consuming work to implement, so it will be nice to have as a standard feature in the R3-GUI.

2. Ideas

2.1. VALIDATE-FACE

1. validation occurs through a single function: VALIDATE-FACE <face> and is recursive, can be done on window level or face level.

2. validation of single faces with an INPUT tag

3. validation of panels of faces with INPUT tags

2.2. Validation indication

1. validation indicator icon that would have to reside to the right of the face that needs to be validated. this is a bit informal to do it like that, but the setup is common and would be flexible enough to use both on single faces and entire panels.

2. validation indicator is not obligatory to attach to a face

3. method for triggering validation for a single face. not sure a reactor is useful. result of validation is logic! where true is 'valid

2.3. Validation states

1. indicator states: 'not-required, 'required, 'valid, 'invalid
2. 'required means that when validation fails, this is a fatal error
3. 'not-required means that when validation fails, this is not a fatal error
4. keyword for indication of 'required field in layout
5. turn off validation on disabled faces

2.4. Validation modes

init	sets all valid-indicators to initial state: 'valid, 'required or 'not-required, would be appropriate for reopened filled forms and cleared new forms
preliminary	performs validation, but doesn't show result - useful for when validation is required dyring typing in form
submission	validate and show all results and auto-focus first invalid field - useful on final validation when wanting to submit

1. state transition on init: valid # valid, invalid # required, not-required # not-required, required # required
2. state transition on validate: not-required # valid, required # valid or invalid, valid # valid, invalid # invalid

2.5. When to validate

1. initial validation on window open. this would work both for new forms and already filled forms as indicated by the state transition above.
2. submission validation on window submission-close, when clicking a submit button
3. clear validation result on window cancel-close, when clicking a cancel button
4. sum of validation for entire form returns logic!
5. need a VALIDATION word in each face that needs validation. it holds the state words: 'not-required, 'required, 'valid, 'invalid or NONE, if the face is not to be validated at all.
6. function to collect a list of faces that are invalid by traversing the GUI for the VALIDATION word. name is TBD.
7. buttons that perform validation, such as a window submit button.

3. Source examples

This would validate the field, when clicking the button:

```
view [
        f: field validate [not empty? get-face face] valid-indicator
        button "Validate" do [validate-face f]
]
```

4. Notes

If this is done right, the only code the user needs to write is the code to validate the field, and otherwise place indicators and a submit button. The GUI handles the rest.

I wouldn’t want to provide standard validation code through keywords. The issue is that this splits the implementation in a need for custom validation and one that follows standard keywords and that’s too messy.

It’s possible that we need to rework it to fit better with the philosophy of how to attach meta-information to a layout stated in the db-reactors.r3 file.

I know this validation scheme is possible, because I’ve already implemented around 75% of it in the VID Extension Kit.