

# Model Validation

Status and future goodness

Honza Kral  
honza.kral@gmail.com

# Current state

- Only `Forms` can currently validate themselves
- `ModelForms` respect some constraints of `Fields`
- No way to validate models
- DB errors are propagated through `.save()`

# Objective

Bring validation into the models

- Have forms respect this
- Don't duplicate code already in forms
- Be as backwards compatible as possible

# Validators

- Centralize all the logic in validators

```
from django.core.exceptions import ValidationError

def is_weirdly_logical(value, all_values={},
                      model_instance=None):
    if model_instance and not model_instance.likes(value):
        raise ValidationError('Women and children first.')
    elif value in all_values:
        raise ValidationError('Weird logic FAIL.')
```

- Have Forms and Models use those

# Callable objects for more flexibility

```
class RequiredIfOtherFieldNotGiven:
    def __init__(self, field):
        self.field = field

    def __call__(self, value, all_values={},
                 model_instance=None):
        if get_value(all_values,
                     model_instance,
                     self.field):
            ...
```

```
SomeField(validators=[Required...Given('other')])
```

# Hooks on (Db)Field

- `validators` arg to `__init__`
  - List of validators to run
- `.clean()`
  - **Calls** `to_python()` **and** `validate()`
- `.validate()`
  - Performs validation (blank, null, ...)
  - Runs validators

# Hooks on Model

- `.clean()`
  - **Calls** `clean()` on all fields
- `.validate()`
  - **Calls** `validate_unique()` (cheers Alex)
  - **Place** for custom validation logic

# Hooks on (Form)Field

- `validators` arg to `__init__`
  - List of validators to run
- `default_validators`
  - List of validators to use. Should replace most of the logic in Fields.

# Custom messages

- Not implemented yet
- On `ValidationError`
  - Error code (constant) passed in along with the default message and params
- Ability to supply an error message dict to `ValidationError` before outputting

# Changes to ModelForm

- `self.instance` **always** being around
  - created in `.clean()`
  - `.clean()` **runs** `self.instance.clean()`
  - errors are being propagated to individual fields
- `validate_unique` **moved to Model** where it belongs

# Changes to FormSet(s)

- All formsets call `form.save()` for all their saving
- `save_as_new` became hackish
  - how to validate something without required field

# Changes to Admin

- Admin now has to save the model before validating `edit_inline`
  - `model.delete()` **or** `transaction.rollback()` if form not valid

# Backwards Incompatible Changes

- Bug fixed:

```
class MyModelForm(ModelForm):  
    class Meta:  
        model = MyModel  
        exclude = ('required_field', )
```

- MyModelForm **will NEVER** validate unless you **prefill** `required_field`

# GSOC

- Thanks Google :-)
- Joseph Kocherhans is my mentor
- Look forward to weekly updates

# History

- Started work at PyCon 2008
  - 80% done
- Continued at EuroPython 2008
- Almost got in before 1.0
- Still only 90% done

# State of work

90% done

- that doesn't mean much
- “only“ custom messages left to design
- we might revisit most design decisions
- and we will revisit naming
- all tests pass, you can specify model validation
- remaining 10% is the „complicated stuff“ and major cleanup (including docs and tests)

# Where to see?

- Github honzakral/django branch model-validation
  - <http://github.com/HonzaKral/django/tree/model-validation>
- GSOC SVN once it's established

# How to prepare

- Define your own `.validate()`
  - Have it raise `forms.ValidationError` for now
  - Place logic in simple functions (validators)
  - Call that from your Form
- Use my branch :-)

# ValidatingForm

```
class ValidatingForm(ModelForm):
    def clean(self):
        s = super(ValidatingForm, self)
        data = s.clean()
        self.instance = s.save(commit=False)
        self.instance.clean()
        return data

    def save(self, commit=True):
        if commit:
            self.instance.save()
            self.save_m2m()
        return self.instance
```

How to help

**TEST**

**Q & (some) A**