django-filemaker Documentation

Release 0.2.2

Luke Pomfrey

April 29, 2016

Contents

Pythonic FileMaker® access and FileMaker layout to Django model mapping.

`django-filemaker` provides a framework for basic interaction with a FileMaker® database via its web XML publisher interface, and a simple model style formulation to retrieve objects from FileMaker® and map them into Django model instances.

CHAPTER 1

Quickstart

Create a `FileMakerModel`:

```
from django.contrib.flatpages.models import FlatPage
from django.contrib.sites.models import Site
from filemaker import fields, FileMakerModel


class FileMakerFlatPage(FileMakerModel):

    # The first argument to a FileMaker field should be the field name for
    # that item on the FileMaker layout
    pk = fields.IntegerField('zpkFlatpageID')
    url = fields.CharField('Url_FileMaker_Field')
    title = fields.CharField('Title_FileMaker_Field')
    content = fields.CharField('Content_FileMaker_Field')
    # You can pass in a default value to any field
    template_name = fields.CharField(
        'Template_Name_Field', default='flatpages/default.html')
    registration_required = fields.BooleanField(
        'Registration_Required_Field', default=False)
    sites = fields.ModelListField('SITES', model=FileMakerSite)

    meta = {
        'connection': {
            'url': 'http://user:password@example.com/',
            'db': 'Db_Name',
            'layout': 'Layout_Name',
        },
        'model': FlatPage,
        'pk_name': 'pk',
    }

class FileMakerSite(FileMakerModel):
    # On related fields we specify the relative field to the field name
    # specified on the calling model (FileMakerFlatPage), unless the
    # calling model uses the special '+self' value which passes the layout
    # of that model to the sub model
    domain = fields.CharField('Domain_field')
    name = fields.CharField('Name_Field')

    meta = {
        'model': Site,
        # abstract here means it is a child of an actual FileMaker layout
```

```
        'abstract': True,
    }
```

Query FileMaker for instances of your model, and convert them to django instances using the `to_django` method:

```
>>> # The Django style methods will convert field names
>>> FlatPage.objects.count() == 0
True
>>> fm_page = FileMakerFlatPage.objects.get(pk=1)
>>> fm_page.to_django()
<FlatPage: pk=1>
>>> FlatPage.objects.count() == 1
True
```

You can also use the FileMaker style manager methods to query FileMaker, these return the result as a *filemaker.parser.FMXMLObject*:

```
>>> FileMakerFlatPage.objects.find(zpkFlatpageID=1)
<FMXMLObject: ...>
```

CHAPTER 2

Contents

2.1 Managers

2.1.1 The raw FileMaker manager

class `filemaker.manager.`**`RawManager`**(*url*, *db*, *layout*, *response_layout=None*, ***kwargs*)

The raw manager allows you to query the FileMaker web interface.

Most manager methods (the exceptions being the committing methods; `find`, `find_all`, `edit`, `new`, and `delete`) are chainable, enabling usage like

```
manager = RawManager(...)
manager = manager.filter(field=value).add_sort_param('some_field')
results = manager.find_all()
```

`__init__`(*url*, *db*, *layout*, *response_layout=None*, ***kwargs*)

Parameters

- **`url`** – The URL to access the FileMaker server. This should contain any authorization credentials. If a path is not provided (e.g. no trailing slash, like `http://username:password@192.168.1.2`) then the default path of `/fmi/xml/fmresultset.xml` will be used.
- **`db`** – The database name to access (sets the `-db` parameter).
- **`layout`** – The layout to use (sets the `-lay` parameter).
- **`response_layout`** – (*Optional*) The layout to use (sets the `-lay.response` parameter).

`add_db_param`(*field*, *value*, *op=None*)

Adds an arbitrary parameter to the query to be performed. An optional operator parameter may be specified which will add an additional field to the parameters. e.g. `.add_db_param('foo', 'bar')` sets the parameter `...&foo=bar&...`, `.add_db_param('foo', 'bar', 'gt')` sets `...&foo=bar&foo.op=gt&...`.

Parameters

- **`field`** – The field to query on.
- **`value`** – The query value.
- **`op`** – (*Optional*) The operator to use for this query.

add_sort_param(*field*, *order=u'ascend'*, *priority=0*)
Add a sort field to the query.

Parameters

- **field** – The field to sort on.
- **order** – (*Optional*, defaults to `ascend`) The direction to sort, one of `ascending` or `descending`.
- **priority** – (*Optional*, defaults to `0`) the order to apply this sort in if multiple sort fields are specified.

delete(***kwargs*)
Deletes a record using the `-delete` command. This method internally calls `_commit` and is not chainable.

You should have either called the *set_record_id()* and/or *set_modifier_id()* methods on the manager, or passed in `RECORDID` or `MODID` as params.

Parameters ****kwargs** – Any additional parameters to pass into the URL.

Return type *filemaker.parser.FMXMLObject*

edit(***kwargs*)
Updates a record using the `-edit` command. This method internally calls `_commit` and is not chainable.

You should have either called the *set_record_id()* and/or *set_modifier_id()* methods on the manager, or passed in `RECORDID` or `MODID` as params.

Parameters ****kwargs** – Any additional parameters to pass into the URL.

Return type *filemaker.parser.FMXMLObject*

find(***kwargs*)
Performs the -find command. This method internally calls `_commit` and is not chainable.

Parameters ****kwargs** – Any additional fields to search on, which will be passed directly into the URL parameters.

Return type *filemaker.parser.FMXMLObject*

find_all(***kwargs*)
Performs the -findall command to return all records. This method internally calls `_commit` and is not chainable.

Parameters ****kwargs** – Any additional URL parameters.

Return type *filemaker.parser.FMXMLObject*

new(***kwargs*)
Creates a new record using the `-new` command. This method internally calls `_commit` and is not chainable.

Parameters ****kwargs** – Any additional parameters to pass into the URL.

Return type *filemaker.parser.FMXMLObject*

set_group_size(*max*)
Set the group size to return from FileMaker using the `-max`.

This is defaulted to 50 when the manager is initialized.

Parameters **max** (*integer*) – The number of records to return.

set_logical_operator(*op*)
Set the logical operator to be used for this query using the `-op` parameter.

Parameters **op** – Must be one of `and` or `or`.

set_modifier_id(*modid*)
Sets the `-modid` parameter.

Parameters **modid** – The modifier ID to set.

set_record_id(*recid*)
Sets the `-recid` parameter.

Parameters **recid** – The record ID to set.

set_script(*name*, *option=None*)
Sets the name of the filemaker script to use

Parameters

- **name** – The name of the script to use.
- **option** – (*Optional*) Can be one of `presort` or `prefind`.

set_skip_records(*skip*)
The number of records to skip when retrieving records from FileMaker using the `-skip` parameter.

Parameters **skip** (*integer*) – The number of records to skip.

2.1.2 The `FMXMLObject` response

class `filemaker.parser.`**FMXMLObject**(*data*)
A python container container for results returned from a FileMaker request.

The following attributes are provided:

data
Contains the raw XML data returned from filemaker.

errorcode
Contains the `errorcode` returned from FileMaker. Note that if this value is not zero when the data is parsed at instantiation, then a *filemaker.exceptions.FileMakerServerError* will be raised.

product
A dictionary containing the FileMaker product details returned from the server.

database
A dictionary containing the FileMaker database information returned from the server.

metadata
A dictionary containing any metadata returned by the FileMaker server.

resultset
A list containing any results returned from the FileMaker server as *FMDocument* instances.

field_names
A list of field names returned by the server.

target
The target class used by lxml to parse the XML response from the server. By default this is an instance of `FMXMLTarget`, but this can be overridden in subclasses.

class `filemaker.parser.`**FMDocument**
A dictionary subclass for containing a FileMaker result whose keys can be accessed as attributes.

2.1.3 The FileMakerModel Manager

class filemaker.manager.**Manager**(*cls*)

A manager for use with *filemaker.base.FileMakerModel* classes. Inherits from the *RawManager*, but adds some conveniences and field mapping methods for use with *filemaker.base.FileMakerModel* sub-classes.

This manager can be treated as an iterator returning instances of the relavent *filemaker.base.FileMakerModel* sub-class returned from the FileMaker server. It also supports slicing etc., although negative indexing is unsupported.

__init__(*cls*)

Parameters **cls** – The *filemaker.base.FileMakerModel* sub-class to use this manager with. It is expected that the model `meta` dictionary will have a `connection` key to a dictionary with values for `url`, `db`, and `layout`.

all()

A no-op returning a clone of the current manager

count()

Returns the number of results returned from FileMaker for this query.

filter(***kwargs*)

Filter the queryset by model fields. Model field names are passed in as arguments rather than FileMaker fields.

Queries spanning relationships can be made using a ___, and operators can be specified at the end of the query. e.g. Given a model:

```
class Foo(FileMakerModel):
    beans = fields.IntegerField('FM_Beans')

    meta = {
        'abstract': True,
        ...
    }

class Bar(FileMakerModel):
    foo = fields.ModelField('BAR_Foo', model=Foo)
    num = models.IntegerField('FM_Num'))

    meta = {
        'connection': {...},
        ...
    }
```

To find all instances of a `Bar` with `num == 4`:

```
Bar.objects.filter(num=4)
```

To find all instances of `Bar` with `num < 4`:

```
Bar.objects.filter(num__lt=4)
```

To Find all instance of `Bar` with a `Foo` with `beans == 4`:

```
Bar.objects.filter(foo__beans=4)
```

To Find all instance of `Bar` with a `Foo` with `beans > 4`:

```
Bar.objects.filter(foo__beans__gt=4)
```

The `filter` method is also chainable so you can do:

```
Bar.objects.filter(num=4).filter(foo__beans=4)
```

Parameters ****kwargs** – The fields and values to filter on.

get(***kwargs*)
Returns the first item found by filtering the queryset by `**kwargs`. Will raise the `DoesNotExist` exception on the managers model class if no items are found, however, unlike the Django ORM, will silently return the first result if multiple results are found.

Parameters ****kwargs** – Field and value queries to be passed to *`filter()`*

order_by(**args*)
Add an ordering to the queryset with respect to a field.

If the field name is prepended by a − that field will be sorted in reverse. Multiple fields can be specified.

This method is also chainable so you can do, e.g.:

```
Foo.objects.filter(foo='bar').order_by('qux').filter(baz=1)
```

Parameters ***args** – The field names to order by.

preprocess_resultset(*resultset*)
This is a hook you can override on a manager to pre-process a resultset from FileMaker before the data is converted into model instances.

Parameters **resultset** – The `resultset` attribute of the *`filemaker.parser.FMXMLObject`* returned from FileMaker

2.2 FileMakerModels

class `filemaker.base.`**`FileMakerModel`**
`FileMakerModel` objects provide a way to map FileMaker layouts to Django models, providing validation, and query methods.

They provide a simple field based API similar to the Django model interface.

to_dict()
The *`to_dict()`* method serializes the FileMaker model hierarchy represented by this model instance to a dictionary structure.

to_django()
The *`to_django()`* converts this FileMaker model instance into an instance of the Django model specified by the `model` value of the classes *`meta`* dictionary.

meta
the *`meta`* dictionary on a FileMaker model class is similar to the `Meta` class on a Django model. See *The meta dictionary*, below, for a full list of options.

2.2.1 The `meta` dictionary

The `meta` dictionary on a model is equivalent to the `Meta` class on a Django model. The `meta` dictionary may contain any of the following keys.

`connection`: For the base model to be queried from a FileMaker layout, this should contain a `connection` dictionary with `url`, `db`, and `layout` fields (and an optional `response_layout` field).

`model`: The Django model class that this *`FileMakerModel`* maps to.

`pk_name`: The field name on the *`FileMakerModel`* that maps to the Django model `pk` field. By default this is `id` or `pk` whichever field is present.

`django_pk_name`: The django `pk` field name. This should almost never need to be changed unless you're doing (*very*) weird things with your Django models.

`django_field_map`: An optional dictionary mapping fields on the *`FileMakerModel`* to fields on the Django model. By default the names are mapped one-to-one between the two.

`abstract`: If this is set to `True` it denotes that the model is a subsection of the layout fields, or a list-field on the layout, i.e. this model doesn't specify a `connection` but is connected to one that does by one or more *`filemaker.fields.ModelField`* or *`filemaker.fields.ModelListField`* instances.

`to_many_action`: If this is set to `clear`, the default, then when converting *`FileMakerModel`* instances to Django instances, existing many-to-many relations will be cleared before being re-added.

`ordering`: Does what it says on the tin. `id` by default.

`default_manager`: The default manager class to use for the model. This is *`filemaker.manager.Manager`* by default.

`related` and `many_related`: These contain reverse entries for *`filemaker.fields.ModelField`* or *`filemaker.fields.ModelListField`* instances on other models pointing back to the current model.

2.2.2 Declaring fields

Fields are declared exactly as with Django models, the exception being that FileMaker fields are used. Field names should either have the same name as their Django model counterparts, unless you are using the `django_field_map` attribute of the `meta` dictionary. For example, we could write a FileMakerModel mapping to the Django FlatPage model as the following:

```
from django.contrib.flatpages.models import FlatPage
from django.contrib.sites.models import Site
from filemaker import FileMakerModel, fields

class FileMakerSite(FileMakerModel):
    d = fields.CharField('FM_domain')
    n = fields.CharField('FM_name')

    meta = {
        'model': Site,
        'abstract': True,
        'django_field_map': {
            'd': 'domain',
            'n': 'name',
        },
    }

class FileMakerFlatPage(FileMakerModel):
```

```
    url = fields.CharField('FM_url')
    title = fields.CharField('FM_title')
    content = fields.CharField('FM_content', default='')
    enable_comments = fields.BooleanField('FM_enable_comments')
    template_name = fields.CharField('FM_template_name')
    registration_required = fields.BooleanField('FM_registration_required')
    sites = fields.ModelListField('SITES', model=FileMakerSite)

    meta = {
        'connection': {
            'url': 'http://user:pass@192.168.0.2',
            'db': 'main',
            'layout': 'flatpages',
        },
        'model': FlatPage,
    }
```

Here we have used different field names on the `FileMakerSite` model, and re-mapped them to the Django `Site`. We have here assumed a FileMaker layout structure something like:

```
- FM_url: The URL for the flatpage.
  FM_title: The title of the flatpage.
  FM_content: The content for the flatpage.
  FM_enable_comments: ...
  FM_template_name: ...
  FM_registration_required: ...
  SITES:
      - FM_domain: The domain of the site.
        FM_name: The name of the site
      - FM_domain: ...
        FM_name: ...
      - ...
- FM_url: ...
  FM_title: ...
  ...
- ...
```

For a full list of field type see the *FileMakerField reference*.

2.2.3 Instantiating a model instance

Models can be instantiated from a FileMaker result by passing a *filemaker.parser.FMDocument* from the *filemaker.parser.FMXMLObject.resultset* attribute of an *filemaker.parser.FMXMLObject* as the first argument. This is how the *filemaker.manager.Manager.get()* and *filemaker.manager.Manager.filter()* methods generate a list of objects internally.

Alternatively you can construct an instance by passing any number of fields names as keyword arguments. So for our `FileMakerFlatPage` above we could do:

```
>>> flat_page = FileMakerFlatPage(
        enable_comments=False, registration_required=False, url='/')
>>> flat_page.title = 'Home'
>>> flat_page.content = 'Testing, testing, 1, 2, 3.'
...
```

Validation is performed as fields are set, e.g.:

```
>>> flat_page = FileMakerFlatPage(sites='i-should-be-a-list')
FileMakerValidationError: ...
```

2.3 Fields

Fields provide the data coersion and validation when pulling data from FileMaker. All fields should inherit from the `BaseFileMakerField`.

class `filemaker.fields.`**`BaseFileMakerField`**(*fm_attr=None*, **args*, ***kwargs*)

This is the base implementation of a field. It should not be used directly, but should be inherited by every FileMaker field class.

Parameters

- **`fm_attr`** – The attribute on the FileMaker layout that this field relates to. If none is given then the field name used on the FileMakerModel will be substituted.
- **`**kwargs`** – And keyword arguments are attached as attributes to the field instance, or used to override base attributes.

Aside from the `fm_attr` following attributes are defined by the base field:

`null`
Determines whether this field is allowed to take a `None` value.

`null_values`
The values that will be treated as `null`, by default the empty string `''` and `None`.

`default`
The default value to use for this field if none is given.

`validators`
A list of functions that take a single argument that will be used to validate the field. Compatible with Django's field validators.

`min`
Specifies the minimum value this field can take.

`max`
Specifies the maximum value this field can take.

`coerce`(*self*, *value*)
Takes a value and either returns the value coerced into the required type for the field or raises a *`filemaker.exceptions.FileMakerValidationError`* with an explanatory message. This method is called internally by the private `_coerce` method during validation.

`to_django`(*self*, **args*, ***kwargs*)
Does any processing on the fields' value required before it can be passed to a Django model. By default this just returns `self.value`.

The current value of a FileMaker field is available using the `value` attribute.

2.3.1 FileMakerField reference

The following fields are provided by Django filemaker.

class `filemaker.fields.`**`BaseFileMakerField`**(*fm_attr=None*, **args*, ***kwargs*)
The base class that all FileMaker fields should inherit from.

Sub-classes should generally override the coerce method which takes a value and should return it in the appropriate format.

class filemaker.fields.**BooleanField**(*fm_attr=None*, **args*, ***kwargs*)
Coerces data into a boolean.

Parameters **map** – An optional dictionary mapping that maps values to their Boolean counterparts.

class filemaker.fields.**BytesField**(*fm_attr=None*, **args*, ***kwargs*)
Coerces data into a bytestring instance

class filemaker.fields.**CharField**(*fm_attr=None*, **args*, ***kwargs*)
An alias for *UnicodeField*.

class filemaker.fields.**CommaSeparatedIntegerField**(*fm_attr=None*, **args*, ***kwargs*)
A *CharField* that validates a comma separated list of integers

class filemaker.fields.**CommaSeparratedIntegerField**(**args*, ***kwargs*)
Alternate (misspelled) name for *CommaSeparatedIntegerField*

Deprecated since version 0.1.1: This field class is deprecated as of 0.1.1 and will disappear in 0.2.0. Use *CommaSeparatedIntegerField* instead.

class filemaker.fields.**CurrencyField**(*fm_attr=None*, **args*, ***kwargs*)
A decimal field that uses 2 decimal places and strips off any currency symbol from the start of it's input.

Has a default minimum value of `0.00`.

class filemaker.fields.**DateField**(*fm_attr=None*, **args*, ***kwargs*)
Coerces data into a datetime.date instance.

Parameters **strptime** – An optional strptime string to use if falling back to the datetime.datetime.strptime method

class filemaker.fields.**DateTimeField**(*fm_attr=None*, **args*, ***kwargs*)
Coerces data into a datetime.datetime instance.

Parameters **strptime** – An optional strptime string to use if falling back to the datetime.datetime.strptime method

class filemaker.fields.**DecimalField**(*fm_attr=None*, **args*, ***kwargs*)
Coerces data into a decimal.Decimal object.

Parameters **decimal_places** – (*Optional*) The number of decimal places to truncate input to.

class filemaker.fields.**EmailField**(*fm_attr=None*, **args*, ***kwargs*)
A *CharField* that vaidates that it's input is a valid email address.

class filemaker.fields.**FileField**(*fm_attr=None*, **args*, ***kwargs*)
A field that downloads file data (e.g. from the FileMaker web interface). The file will be saved with a filename that is the combination of the hash of it's contents, and the extension associated with the mimetype it was served with.

Can be given an optional `base_url` with which the URL received from FileMaker will be joined.

Parameters

- **retries** – The number of retries to make when downloading the file in case of errors. Defaults to `5`.
- **base_url** – The URL with which to combine the url received from FileMaker, empty by default.
- **storage** – The Django storage class to use when saving the file. Defaults to the default storage class.

class filemaker.fields.**FloatField**(*fm_attr=None*, **args*, ***kwargs*)
Coerces data into a float.

class filemaker.fields.**GTINField**(*fm_attr=None*, **args*, ***kwargs*)
A *CharField* that validates it's input is a valid GTIN.

class filemaker.fields.**IPAddressField**(*fm_attr=None*, **args*, ***kwargs*)
A *CharField* that validates that it's input is a valid IPv4 or IPv6 address.

class filemaker.fields.**IPv4AddressField**(*fm_attr=None*, **args*, ***kwargs*)
A *CharField* that validates that it's input is a valid IPv4 address.

class filemaker.fields.**IPv6AddressField**(*fm_attr=None*, **args*, ***kwargs*)
A *CharField* that validates that it's input is a valid IPv6 address.

class filemaker.fields.**ImageField**(*fm_attr=None*, **args*, ***kwargs*)
A *FileField* that expects the mimetype of the received file to be `image/*`.

class filemaker.fields.**IntegerField**(*fm_attr=None*, **args*, ***kwargs*)
Coerces data into an integer.

class filemaker.fields.**ListField**(*fm_attr=None*, **args*, ***kwargs*)
A field that takes a list of values of other types.

Parameters **base_type** – The base field type to use.

class filemaker.fields.**ModelField**(*fm_attr=None*, **args*, ***kwargs*)
A field that provides a refernce to an instance of another filemaker model, equivalent to a Django ForeignKey.

Parameters **model** – The FileMaker model to reference.

class filemaker.fields.**ModelListField**(*fm_attr=None*, **args*, ***kwargs*)
A fields that gives a reference to a list of models, equivalent to a Django ManyToMany relation.

Parameters **model** – The model class to reference.

class filemaker.fields.**NullBooleanField**(*fm_attr=None*, **args*, ***kwargs*)
A BooleanField that also accepts a null value

class filemaker.fields.**PercentageField**(*fm_attr=None*, **args*, ***kwargs*)
A *DecimalField* that ensures it's input is between 0 and 100

class filemaker.fields.**PositiveIntegerField**(*fm_attr=None*, **args*, ***kwargs*)
An *IntegerField* that ensures it's input is 0 or greater.

class filemaker.fields.**SlugField**(*fm_attr=None*, **args*, ***kwargs*)
A *CharField* that validates it's input is a valid slug. Will automatically slugify it's input, by default. Can also be passed a specific slugify function.

Note: If the custom `slugify` function would create a slug that would fail a test by `django.core.validators.validate_slug` it may be wise to pass in a different or empty `validators` list.

Parameters

- **auto** – Whether to slugify input. Defaults to `True`.
- **slugify** – The slugify function to use. Defaults to `django.template.defaultfilters.slugify`.

class filemaker.fields.**TextField**(*fm_attr=None*, **args*, ***kwargs*)
An alias for *UnicodeField*.

class filemaker.fields.**ToManyField**(*fm_attr=None*, **args*, ***kwargs*)
An alias for *ModelListField*.

class filemaker.fields.**ToOneField**(*fm_attr=None*, **args*, ***kwargs*)
An alias for *ModelField*

class filemaker.fields.**URLField**(*fm_attr=None*, **args*, ***kwargs*)
A *CharField* that validates it's input is a valid URL.

class filemaker.fields.**UnicodeField**(*fm_attr=None*, **args*, ***kwargs*)
Coerces data into a `unicode` object on Python 2.x or a `str` object on Python 3.x

class filemaker.fields.**UploadedFileField**(*fm_attr=None*, **args*, ***kwargs*)
Takes the path of a file that has already been uploaded to storage and generates a File instance from it.

Parameters **storage** – An instance of the storage class to use

2.3.2 Creating custom fields

If your custom field only requires a simple validation check, then it is easiest to override the validators list for a field by passing in a new list of validators.

If you require more control over your field, you can subclass *BaseFileMakerField*, or the field class that most closely resembles your desired type. The two methods that you will likely wish to overwrite are the *BaseFileMakerField.coerce()* method, and the *BaseFileMakerField.to_django()* method.

BaseFileMakerField.coerce() is called by the private `_coerce` method during validation. It should take a single `value` parameter and either return an instance of the required type, or raise a `filemake.exceptions.FileMakerValidationError` with an explanation of why the value could not be coerced.

BaseFileMakerField.to_django() does any post processing on the field required to render it suitable for passing into a Django field. By default this method just returns the field instances' current value.

As an example, if we wanted to have a field that took a string and added "from FileMaker" to the end of it's value we could do:

```
from filemaker.fields import CharField

class FromFileMakerCharField(CharField):

    def coerce(self, value):
        text = super(FromFileMakerCharField, self).coerce(value)
        return '{0} from FileMaker'.format(text)
```

If we wanted to remove the extra string before passing the value into a Django model, we could add a *BaseFileMakerField.to_django()* method, like so:

```
import re

from filemaker.fields import CharField

class FromFileMakerCharField(CharField):

    def coerce(self, value):
        text = super(FromFileMakerCharField, self).coerce(value)
        return '{0} from FileMaker'.format(text)
```

```
    def to_django(self):
        return re.sub(r' from FileMaker$', '', self.value)
```

2.4 Exceptions

exception `filemaker.exceptions.`**`FileMakerError`**
This is the base exception related to FileMaker operations. All other exceptions here inherit from it.

exception `filemaker.exceptions.`**`FileMakerValidationError`**(**args*, ***kwargs*)
Raised when a FileMaker field fails validation. The same as Django's `django.core.exceptions.ValidationError` (from which it inherits).

When raised by a field, the field raising it will be available on the exception as the `field` attribute.

exception `filemaker.exceptions.`**`FileMakerObjectDoesNotExist`**
Raised when no `FileMakerModel` instance matching a query is found.

Every `FileMakerModel` has a sub-class of this as *DoesNotExist* enabling you to catch exceptions raised by a specific model.

exception `filemaker.exceptions.`**`FileMakerConnectionError`**
Raised when an HTTP or other error is encountered when attempting to connect to the FileMaker server.

exception `filemaker.exceptions.`**`FileMakerServerError`**(*code=-1*)
Indicates an error returned by FileMaker. This is raised by the result parser and in turn by the FileMaker managers.

The exact FileMaker error code is available on the exception as the `code` attribute, and the corresponding text representation of the error is on the `message` attribute.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

f

Symbols

A

B

C

D

E

F

G

I

L

M

N

O

P

R

S

T

U

V