plone.app.fhirfield Documentation

Release 1.0.0rc1

Md Nazrul Islam

Nov 12, 2019

Contents

1 Introduction 1

2 FHIR Search Query 3

3 API Documentation 5

4 RESTfull API Service 7

5 Migration 15

6 Changelog 17

7 Contribute 23

8 Contributors 25

9 License 27

10 Credits 29

11 Background (plone.app.fhirfield) 31

12 Available Field's Options 33

13 Field's Value API 35

14 Helper API 37

15 elasticsearch setup 39

16 Installation 41

17 Links 43

18 License 45

19 Indices and tables 47

Python Module Index 49

Index 51

CHAPTER 1

Introduction

FHIR Fast Heathcare Interoperability Resource

CHAPTER 2

FHIR Search Query

This product has a query builder helper, that is actually transforming fhir search params into Plone's search compatible params so that PortalCatalog tool understand what to do. Each of FHIR Indexes provided by this product, by default using this query builder but it is possible to provide prepared query.

2.1 Query Builder

`plone.app.fhir.field.helpers.build_elasticsearch_query` takes only HL7 fhir standard search parameters and transforms to elasticsearch compatible query that is executing through plone catalog search. However belows are lists of standard parameters those are supported by this *Query Builder* (more to continue adding, until full supports are completed)

2.2 Integrate in your REST API service

This product has been got all battery included to be integrated with plone.restapi for becoming HL7 FHIR standard RESTful API server which would provide search service as defined here.

Example RESTful service could be found here

2.3 Supported Search Parameters

Parameter Name	Remarks
_id	Yes
_lastUpdated	Yes
_profile	Yes
birthdate	Yes
gender	Yes

Continued on next page

Table 1 – continued from previous page

Parameter Name	Remarks
patient	Yes
status	Yes
owner	Yes
subject	Yes
effective	Yes
url	Yes
version	Yes
authored	Yes
identifier	Yes
based-on	Yes
part-of	Yes
price-override	Yes
quantity	Yes
factor-override	Yes
length	Yes
code	Yes
medication	Yes
address	Yes
address-city	Yes
address-country	Yes
address-postalcode	Yes
address-state	Yes
address-use	Yes
telecom	Yes
email	Yes
phone	Yes
name	Yes
family	Yes
given	Yes
code-value-concept	Yes
code-value-date	Yes
code-value-quantity	Yes
code-value-string	Yes

CHAPTER 3

API Documentation

3.1 IFhirResource

3.2 IFhirResourceModel

3.3 IFhirResourceValue

CHAPTER 4

RESTfull API Service

4.1 Server Side codex examples

Listing 1: Search aka GET Service

```
# -*- coding: utf-8 -*-
from plone import api
from plone.app.fhirfield.helpers import parse_query_string
from plone.restapi.services import Service
from zope.interface import implementer
from zope.publisher.interfaces import IPublishTraverse


@implementer(IPublishTraverse)
class FHIRSearchService(Service):
    """ """
    def __init__(self, context, request):
        """ """
        super(FHIRSearchService, self).__init__(context, request)
        self.params = []

    def reply(self):
        """ """
        results = [getattr(
            r.getObject(),
            '{0}_resource'.format(self.resource_type).lower()).as_json()
            for r in self.build_result()]

        if self.resource_id:
            if not results:
                self.request.response.setStatus(404)
                return None
            return results[0]
```

(continues on next page)

(continued from previous page)

```
        return results

    def publishTraverse(self, request, name):  # noqa: N802
        # Consume any path segments after /@fhir as parameters
        self.params.append(name)
        return self

    @property
    def resource_id(self):
        """ """
        if 1 < len(self.params):
            return self.params[1]
        return None

    @property
    def resource_type(self):
        """ """

        if 0 < len(self.params):
            _rt = self.params[0]
            return _rt
        return None

    def _get_fhir_fieldname(self, resource_type=None):
        """We assume FHIR Field name is ``{resource type}_resource``"""
        resource_type = resource_type or self.resource_type

        return '{0}_resource'.format(resource_type.lower())

    def get_params(self):
        """We are not using self.request.form (parsed by Zope Publisher)!!
        There is special meaning for colon(:) in key field. For example `field_
↪name:list`
        treats data as List and it doesn't recognize FHIR search modifier like :not,␣
↪:missing
        as a result, from colon(:) all chars are ommited.
        """
        if self.resource_id:
            return {'_id': self.resource_id}

        return parse_query_string(self.request)

    def build_query(self):
        """ """
        query = dict()

        fhir_query = self.get_params()
        extra_params = dict()
        # not supporting count yet!
        if '_count' in fhir_query:
            extra_params['_count'] = fhir_query.pop('_count')

        if 'search-offset' in fhir_query:
            extra_params['search-offset'] = fhir_query.pop('search-offset')

        if 'search-id' in fhir_query:
            extra_params['search-id'] = fhir_query.pop('search-id')
```

(continues on next page)

(continued from previous page)

```
        if fhir_query:
            query[self._get_fhir_fieldname(self.resource_type)] = \
                fhir_query

        return query, extra_params

    def build_result(self):
        """ """
        query, extra_params = self.build_query()
        results = api.content.find(**query)  # noqa: P001

        return results
```

Listing 2: FHIR Resource Add aka POST Service

```
# -*- coding: utf-8 -*-
from Acquisition import aq_base
from Acquisition.interfaces import IAcquirer
from plone import api
from plone.restapi.deserializer import json_body
from plone.restapi.exceptions import DeserializationError
from plone.restapi.interfaces import IDeserializeFromJson
from plone.restapi.services import Service
from plone.restapi.services.content.utils import add as add_obj
from plone.restapi.services.content.utils import create as create_obj
from Products.CMFPlone.utils import safe_hasattr
from zope.component import queryMultiAdapter
from zope.event import notify
from zope.interface import alsoProvides
from zope.interface import implementer
from zope.lifecycleevent import ObjectCreatedEvent
from zope.publisher.interfaces import IPublishTraverse

import json
import plone.protect.interfaces


__author__ = 'Md Nazrul Islam <nazrul@zitelab.dk>'


@implementer(IPublishTraverse)
class FHIRResourceAdd(Service):
    """Creates a new FHIR Resource object.
    """
    def __init__(self, context, request):
        """ """
        super(FHIRResourceAdd, self).__init__(context, request)
        self.params = []

    def publishTraverse(self, request, name):  # noqa: N802
        # Consume any path segments after /@fhir as parameters
        self.params.append(name)
        return self

    @property
```

(continues on next page)

(continued from previous page)

```
    def resource_type(self):
        """ """

        if 0 < len(self.params):
            _rt = self.params[0]
            return _rt
        return None

    def reply(self):
        """ """
        data = json_body(self.request)

        # Disable CSRF protection
        if 'IDisableCSRFProtection' in dir(plone.protect.interfaces):
            alsoProvides(self.request,
                         plone.protect.interfaces.IDisableCSRFProtection)

        response = self._create_object(data)

        if 'error' in response:
            self.request.response.setStatus(400)

        return response

    def _create_object(self, fhir):
        """ """
        form_data = {
            '@type': 'FF' + fhir['resourceType'],
            'id': fhir['id'],
            'title': '{0}-{1}'.format(
                self.resource_type,
                fhir['id'])
        }
        fhir_field_name = '{0}_resource'.format(
            fhir['resourceType'].lower())
        form_data[fhir_field_name] = fhir

        self.request['BODY'] = json.dumps(form_data)

        context = api.portal.get()

        obj = create_obj(
            context,
            form_data['@type'],
            id_=form_data['id'],
            title=form_data['title'])

        if isinstance(obj, dict) and 'error' in obj:
            self.request.response.setStatus(400)
            return obj

        # Acquisition wrap temporarily to satisfy things like vocabularies
        # depending on tools
        temporarily_wrapped = False
        if IAcquirer.providedBy(obj) and not safe_hasattr(obj, 'aq_base'):
            obj = obj.__of__(context)
            temporarily_wrapped = True
```

(continues on next page)

(continued from previous page)

```
        # Update fields
        deserializer = queryMultiAdapter(
            (obj, self.request),
            IDeserializeFromJson
        )
        if deserializer is None:
            self.request.response.setStatus(501)
            return dict(error=dict(
                message='Cannot deserialize type {0}'.format(obj.portal_type)))

        try:
            deserializer(validate_all=True, create=True)
        except DeserializationError as e:
            self.request.response.setStatus(400)
            return dict(error=dict(type='DeserializationError', message=str(e)))

        if temporarily_wrapped:
            obj = aq_base(obj)

        # Notify Dexterity Created
        if not getattr(deserializer, 'notifies_create', False):
            notify(ObjectCreatedEvent(obj))

        # Adding to Container
        add_obj(context, obj, rename=False)

        self.request.response.setStatus(201)
        response = getattr(obj, fhir_field_name).as_json()

        self.request.response.setHeader(
            'Location',
            '/'.join([self.context.portal_url(),
                      '@fhir',
                       response['resourceType'],
                       response['id']]))

        return response
```

Listing 3: FHIR Resource Update aka PATCH Service

```
# -*- coding: utf-8 -*-
from plone import api
from plone.restapi.deserializer import json_body
from plone.restapi.services import Service
from plone.restapi.services.locking.locking import is_locked
from zope.interface import alsoProvides
from zope.interface import implementer
from zope.publisher.interfaces import IPublishTraverse

import plone.protect.interfaces


@implementer(IPublishTraverse)
class FHIRResourcePatch(Service):
    """Patch a FHIR Resource object.
```

(continues on next page)

(continued from previous page)

```
    """
    def __init__(self, context, request):
        """ """
        super(FHIRResourcePatch, self).__init__(context, request)
        self.params = []

    def publishTraverse(self, request, name):  # noqa: N802
        # Consume any path segments after /@fhir as parameters
        self.params.append(name)
        return self

    @property
    def resource_id(self):
        """ """
        if 1 < len(self.params):
            return self.params[1]
        return None

    @property
    def resource_type(self):
        """ """

        if 0 < len(self.params):
            _rt = self.params[0]
            return _rt
        return None

    def reply(self):
        """ """
        query = {
            '{0}_resource'.format(
                self.resource_type.lower()
                ): {'_id': self.resource_id}
            }
        brains = api.content.find(**query)

        if len(brains) == 0:
            self.request.response.setStatus(404)
            return None

        obj = brains[0].getObject()

        if is_locked(obj, self.request):
                self.request.response.setStatus(403)
                return dict(error=dict(
                    type='Forbidden', message='Resource is locked.'))

        data = json_body(self.request)

        # Disable CSRF protection
        if 'IDisableCSRFProtection' in dir(plone.protect.interfaces):
            alsoProvides(self.request,
                         plone.protect.interfaces.IDisableCSRFProtection)

        fhir_value = getattr(
            obj,
            '{0}_resource'.format(self.resource_type.lower()))
```

(continues on next page)

(continued from previous page)

```
        fhir_value.patch(data['patch'])

        self.request.response.setStatus(204)
        # Return None
        return None
```

Listing 4: REST Service registration (configuration.zcml)

```
<configure
    xmlns="http://namespaces.zope.org/zope"
    xmlns:plone="http://namespaces.plone.org/plone"
    xmlns:zcml="http://namespaces.zope.org/zcml">


    <include package="plone.rest" file="configure.zcml" />
    <plone:service  method="GET"
                    name="@fhir"
                    for="Products.CMFCore.interfaces.ISiteRoot"
                    factory=".get.FHIRSearchService"
                    permission="zope2.View" />

    <plone:service  method="POST"
                    name="@fhir"
                    for="Products.CMFCore.interfaces.ISiteRoot"
                    factory=".post.FHIRResourceAdd"
                    permission="cmf.ManagePortal" />

    <plone:service  method="PATCH"
                    name="@fhir"
                    for="Products.CMFCore.interfaces.ISiteRoot"
                    factory=".patch.FHIRResourcePatch"
                    permission="cmf.ManagePortal" />

</configure>
```

4.2 REST Client Examples

Getting single resource, here we are getting Patient resource by ID.

Example(1):

```
>>> response = admin_session.get('/@fhir/Patient/19c5245f-89a8-49f8-b244-666b32adb92e
↪')
>>> response.status_code
200

>>> response.json()['resourceType'] == 'Patient'
True

>>> response = admin_session.get('/@fhir/Patient/19c5245f-fake-id')
>>> response.status_code
404
```

Search Observation by Patient reference with status condition. Any observation until December 2017 and earlier than January 2017.

Example(2):

```
>>> response = admin_session.get('/@fhir/Observation?patient=Patient/19c5245f-89a8-
↪49f8-b244-666b32adb92e&status=final&_lastUpdated=lt2017-12-31&_lastUpdated=gt2017-
↪01-01')
>>> response.status_code
200
>>> len(response.json())
1
```

Add FHIR Resource through REST API

Example(3):

```
>>> import os
>>> import json
>>> import uuid
>>> import DateTime
>>> import time

>>> with open(os.path.join(FIXTURE_PATH, 'Patient.json'), 'r') as f:
...     fhir_json = json.load(f)

>>> fhir_json['id'] = str(uuid.uuid4())
>>> fhir_json['name'][0]['text'] = 'Another Patient'
>>> response = admin_session.post('/@fhir/Patient', json=fhir_json)
>>> response.status_code
201
>>> time.sleep(1)
>>> response = admin_session.get('/@fhir/Patient?active=true')
>>> len(response.json())
2
```

Update (PATCH) FHIR Resource the Patient is currently activated, we will deactive.

Example(4):

```
>>> patch = [{'op': 'replace', 'path': '/active', 'value': False}]
>>> response = admin_session.patch('/@fhir/Patient/19c5245f-89a8-49f8-b244-
↪666b32adb92e', json={'patch': patch})
>>> response.status_code
204
```

CHAPTER 5

Migration

5.1 1.0.0 to 2.0.0

1. Simply old indexes are not usable any more! as new ES server (6.x.x) is used.
2. Do backup of your existing Data.fs (important!).
3. From plone controlpanel, Go to elasticsearch settings page `/@@elastic-controlpanel`.
4. Convert Catalog again, make sure in the `Indexes for which all searches are done through ElasticSearch` section your desired indexes are select.
5. If your site is behind the proxy, we suggest make ssh tuneling to connect your site. Because next takes much times, (depends on your data size)
6. Rebuild Catalog && wait for success, if you face any problem try again.

5.2 1.0.0rc3 to 1.0.0rc4

1. Keep backup of existing Data.fs (nice to have)
2. Go to plone control panel's Addon settings
3. Uninstall and Install again *plone.app.fhirfield*

5.3 1.0.0b6 to 1.0.0rc3

1. Keep backup of existing Data.fs (important)
2. Just *Clear and Rebuild* existing catalogs. Go to site management -> portal_catalog -> Advanced Tab and click the *Clear and Rebuild* button. Caution: this activity could take longer time.

5.4 1.0.0bx to 1.0.0b6

1. You will need to remove all the indexes who are based on *Fhir*******Index*. Go to ZMI to portal catalog tool. *{site url}/portal_catalog/manage_catalogIndexes*.
2. Update the *plone.app.fhirfield* version and run the buildout.
3. List out any products those defined (profile/catalog.xml) indexes based on *Fhir*******Index* as meta type. Reninstall them all.
4. From ZMI, portal_catalog tool, *{site url}/portal_catalog/manage_catalogAdvanced* `Clear and Rebuild`

CHAPTER 6

Changelog

6.1 3.0.0b2 (unreleased)

- Nothing changed yet.

6.2 3.0.0b1 (2019-10-01)

Newfeatures

- FHIRPath support added through collective.fhirpath.
- Now supports Elasticsearch server version `6.8.3`.

Breakings

- Drop support python 2.x.x.

6.3 2.0.0 (2019-05-18)

Newfeatures

- Issue#14 Now reference query is powerful yet! It is possible now search by resourceType only or mixing up.
- Issue#21 One of the powerful feature added, IN/OR support in search query. Now possible to provide multiple values separated by comma.

Breakings

- Drop support Elastic server version 2.3.x.

Bugfixes

- Important fix for `Quantity` search type, now value prefix not impact on other (unit, system). Additionally also now possible to search by unit or system and code (without value)

6.4 1.0.0 (2019-01-18)

Breaking

- Drop `fhirclient` dependency, instead make `fhir.resources` dependency.
- `collective.elasticsearch` version minimum version `2.0.5` has been pinned.

6.5 1.0.0rc4 (2018-10-25)

Breaking

- Drop support for Plone 4.xx (sorry).

Improvements

- Issue#10 JSON Viewer added in display mode.
- Issue#18 *api* module to make available for all publicly usable methods, functions, classes.
- Issue#17 Add suports for duplicate param names into query string. It is now possible to provide multiple condition for same param type. For example *?_lastUpdated=gt2015-10-15T06:31:18+00:00&_lastUpdated=lt2018-01-15T06:31:18+00:00*
- Issue#10 Add support for *Composite* type FHIR search param.
- Issue#13 Add support for *Address* and *ContactPoint* mapping. It opens up many search params.
- Mappings are more enriched, so many missing mappings are added, like *valueString*, *valueQuantity* and so on.[nazrulworld]
- Issue#12 Full support for *code* search param type has been added, it also opens up for other search parameters (y).
- Issue#15 support for *Humanane* mapping in search.

6.6 1.0.0rc3 (2018-09-22)

Improvements

- Issue: 6 A major improvement has been done, now very slim version (*id*, *resourceType*, *meta*) of FHIR json resource has been indexed in ZCatalog (zope index) version, however previously whole fhir resource was stored as a result now huge storage savings, perhaps improves indexing performance. [nazrulworld]

6.7 1.0.0rc2 (2018-08-29)

Bugfixes

- Issue 5: FHIR search's modifier 'missing' is not working for nested mapping

6.8 1.0.0rc1 (2018-08-27)

Newfeatures

- Identifier search parameter is active now (both array of identifier and single object identifier).
- Array of Reference query support (for example *basedOn* (list of reference)) is active now. Although normal object reference has already been supported.
- All available mappings for searchable resources are generated.
- FHIR search sort feature is available!
- *fhirfield.es.index.mapping.nested_fields.limit*, *fhirfield.es.index.mapping.depth.limit* and *fhirfield.es.index.mapping.total_fields.limit* registry records are available to setup ES mapping
- *URI* and *Number* type parameter based search are fully available.
- **'resourceType' filter is automatically injected into every generated query.** Query Builder knows about which resourceType should be.

Breaking Changes

- *plone.app.fhirfield* have to install, as some registry records (settings) for elasticsearch mapping have been introduced.
- Any deprecated FHIR Field Indexes other than *FhirFieldIndex* (*FhirOrganizationIndex* and so on) are removed

6.9 1.0.0b7 (2018-08-10)

- Media search available.
- *plone.app.fhirfield.SearchQueryError* exception class available, it would be used to catch any fhir query buiding errors. [nazrulworld]

6.10 1.0.0b6 (2018-08-04)

- Fix: minor type mistake on non existing method called.
- Migration guide has been added. [nazrulworld]

6.11 1.0.0b5 (2018-08-03)

Newfeatures

- *FhirFieldIndex* Catalog Index has been refactored. Now this class is capable to handle all the FHIR resources. That's why other PluginIndexes related to FhirField have been deprecated.
- New ZCatalog (plone index) index naming convention has been introduced. Any index name for FhirFieldIndex must have fhir resource type name as prefix. for example: *task_index*

6.12 1.0.0b4 (2018-08-01)

- Must Update (fix): Important updates made on mapping, reference field mapping was not working if value contains with /, now made it tokenize by indecating index is *not_analyzed*
- *_profile* search parameter is now available. [nazrulworld]

6.13 1.0.0b3 (2018-07-30)

- Mapping improvment for *FhirQuestionnaireResponseIndex*, *FhirObservationIndex*, *FhirProcedureRequestIndex*, *FhirTaskIndex*, *FhirDeviceRequestIndex*

6.14 1.0.0b2 (2018-07-29)

New Features:

- supports for elasticsearch has been added. Now many basic fhir search are possible to be queried.
- upto 22 FHIR fields indexes (*FhirActivityDefinitionIndex*, *FhirAppointmentIndex*, *FhirCarePlanIndex*, *FhirDeviceIndex*, *FhirDeviceRequestIndex*, *FhirHealthcareServiceIndex*, *FhirMedicationAdministrationIndex*, *FhirMedicationDispenseIndex*, *FhirMedicationRequestIndex*, *FhirMedicationStatementIndex*, *FhirObservationIndex*, *FhirOrganizationIndex*, *FhirPatientIndex*, *FhirPlanDefinitionIndex*, *FhirPractitionerIndex*, *FhirProcedureRequestIndex*, *FhirQuestionnaireIndex*, *FhirQuestionnaireResponseIndex*, *FhirRelatedPersonIndex*, *FhirTaskIndex*, *FhirValueSetIndex*)
- Mappings for all available fhir indexes are created.
- *elasticsearch* option is now available for setup.py

6.15 1.0.0b1 (2018-03-17)

- first beta version has been released.

6.16 1.0.0a10 (2018-03-12)

- fix(bug) Issue-3: *resource_type* constraint don't raise exception from validator.

6.17 1.0.0a9 (2018-03-08)

- There is no restriction/limit over fhir resources, all available models are supported.

6.18 1.0.0a8 (2018-01-22)

- fix(bug) Issue-: Empty string value raise json validation error #2:https://github.com/nazrulworld/plone.app.fhirfield/issues/2

6.19 1.0.0a7 (2018-01-21)

- fix(bug) Issue-1: _RuntimeError: maximum recursion depth exceeded while calling a Python object at form view. #1:https://github.com/nazrulworld/plone.app.fhirfield/issues/1

6.20 1.0.0a6 (2018-01-14)

- missing *HealthcareService* fhir model is added as supported model.

6.21 1.0.0a5 (2018-01-14)

- *Person* fhir model added in whitelist.

6.22 1.0.0a4 (2018-01-14)

- IFhirResource.model_interface field type changed to *DottedName* from *InterfaceField*.

6.23 1.0.0a3 (2017-12-22)

- FHIR Patch support added. Now patching fhir resource is more easy to manage.
- plone.supermodel support is confirmed.[nazrulworld]
- plone.rfc822 marshaler field support.[nazrulworld]

6.24 1.0.0a2 (2017-12-10)

- *FhirResourceWidget* is made working state. From now it is possible to adapt FhirResourceWidget‘ with z3c.form [nazrulworld]

6.25 1.0.0a1 (2017-12-05)

- Initial release. [nazrulworld]

CHAPTER 7

Contribute

7.1 Using the development buildout

Create a virtualenv in the package:

```
$ virtualenv --clear .
```

Install requirements with pip:

```
$ ./bin/pip install -r requirements.txt
```

Run buildout:

```
$ ./bin/buildout
```

Start Plone in foreground:

```
$ ./bin/instance fg
```

CHAPTER 8

Contributors

- Md Nazrul Islam (Author), email2nazrul@gmail.com

CHAPTER 9

License

CHAPTER 10

Credits

10.1 Copyright (third party)

Plone

10.2 Ideas and Concepts

Documentation

Field

CHAPTER 11

Background (plone.app.fhirfield)

FHIR (Fast Healthcare Interoperability Resources) is the industry standard for Healthcare system. Our intend to implement FHIR based system using Plone! plone.app.fhirfield will make life easier to create, manage content for FHIR resources as well search facilities for any FHIR Resources.

11.1 How It Works

This field is working as like other zope.schema field, just add and use it. You will feed the field value as either json string or python dict of FHIR resources through web form or any restapi client. This field has built-in FHIR resource validator and parser.

Example:

```
from plone.app.fhirfield import FhirResource
from plone.supermodel import model

class IMyContent(model.Schema):

    <resource_type>_resource = FhirResource(
        title=u'your title',
        desciption=u'your desciption',
        resource_type='any fhir resource type[optional]'
    )
```

The field's value is the instance of a specilized class *FhirResourceValue* inside the context, which is kind of proxy class of fhirclient model with additional methods and attributes.

Make field indexable

A specilized Catalog PluginIndexes is named `FhirFieldIndex` is available, you will use it as like other catalog indexes. However importantly you have to maintain a strict convention (index name must be started with any valid fhir resource name (no matter uppercase or lowercase) and should _(underscore) be used as separator, i.e task_index(<resource>_<any name>))

Example:

```
<?xml version="1.0"?>
<object name="portal_catalog" meta_type="Plone Catalog Tool">
    <index name="organization_resource" meta_type="FhirFieldIndex">
        <indexed_attr value="organization_resource"/>
    </index>
</object>
```

11.2 Features

- Plone restapi support
- Widget: z3cform support
- plone.supermodel support
- plone.rfc822 marshaler field support
- 100% FHIR search compliance catalog search.

CHAPTER 12

Available Field's Options

This field has got all standard options (i.e *title*, *required*, *desciption* and so on) with additionally options for the purpose of either validation or constraint those are related to FHIR.

resource_type Required: No

Default: None

Type: String

The name of FHIR resource can be used as constraint, meaning that we can restricted only accept certain resource. If FHIR json is provided that contains other resource type, would raise validation error. Example: *FhirResource(. . . .,resource_type='Patient')*

model Required: No

Default: None

Type: String + full python path (dotted) of the model class.

Like *resource_type* option, it can be used as constraint, however additionally this option enable us to use custom model class other than fhirclient's model. Example: *FhirResource(. . . .,model='fhirclient.models.patient.Patient')*

model_interface Required: No

Default: None

Type: String + full python path (dotted) of the model class.

Unlike *model* option, this option has more versatile goal although you can use it for single resource restriction. The advanced usecase like, the field could accept muiltiple resources types those model class implements the provided interface. For example you made a interface called *IMedicalService* and (*Organization*, *Patient*, *Practitioner*) models those are implementing *IMedicalService*. So when you provides this option value, actually three types of fhir json can now be accepted by this field. Example: *FhirResource(. . . .,model='plone.app.interfaces.IFhirResourceModel')*

CHAPTER 13

Field's Value API

Field's value is a specilized class *plone.app.fhirfield.value.FhirResourceValue* which has reponsibilty to act as proxy of fhirclient model's class. This class provides some powerful methods.

FhirResourceValue::as_json

> Originally this method is derived from fhirclient base model, you will always have to use this method during negotiation (although our serializer doing that for you automatically). This method not takes any argument, it returns FHIR json representation of resource.

FhirResourceValue::patch

> If you are familar with FHIRPath Patch, this method one of the strongest weapon of this class. Patch applying on any FHIR resources is noting but so easy. This method takes one mandatory argument *patch_data* and that value should be list of patch items (jsonpatch).
>
> Example:

```
from plone.app.fhirfield import FhirResource
from plone.supermodel import model

class ITask(model.Schema):

    resource = FhirResource(
        title=u'your title',
        desciption=u'your desciption',
        resource_type='Task'
    )

patch_data = [
  {'op': 'replace', 'path': '/source/display', 'value': 'Patched display'},
  {'op': 'replace', 'path': '/status', 'value': 'Reopen'}
]
task_content.resource.patch(patch_data)
```

FhirResourceValue::stringify

This method returns string representation of fhir resource json value. Normally *as_json* returns python's dict type data. This method takes optional *prettify* argument, by setting this argument True, method will return human/print friendly representation.

FhirResourceValue::foreground_origin

There may some situation come, where you will need just pure instance of fhir model, this method serves that purpose. This method returns current fhir resource model's instance.

Example:

```
from fhirclient.models.task import Task
from plone.app.fhirfield import FhirResource
from plone.supermodel import model

class ITask(model.Schema):

    resource = FhirResource(
        title=u'your title',
        desciption=u'your desciption',
        resource_type='Task'
    )

task = task_content.resource.foreground_origin()
assert isinstance(task, Task)
```

CHAPTER 14

Helper API

This package provides some useful functions those could be usable in your codebase.

resource_type_str_to_fhir_model

This function return appropriate fhirclient model class based on provided *resource type*. On wrong resource type *zope.interface.Invalid* exception is raisen.

Example:

```
>>> from plone.app.fhirfield.helpers import resource_type_str_to_fhir_model
>>> task_model_class = resource_type_str_to_fhir_model('Task')
```

CHAPTER 15

elasticsearch setup

If your intent to use elasticsearch based indexing and query, this section for you! you can find more details here

15.1 server setup

server version is restricted to *2.4.x*, means we cannot use latest version of elasticsearch. i.e 5.6.x

- Download from here and install according to documentation.
- For development you could use docker container. The Makefile is available, *~$ make run-es*

15.2 collective.elasticsearch setup

Full configuration guide could be found here. Simple steps are described bellow.

1. **create catalog/indexes**: First you will need add indexes for each fhirfield used in your project. each resource type has it's own Meta Index. example is here
2. Install *collective.elasticsearch* addon from plone control panel.
3. Convert your Indexes to elasticsearch. Go To *{portal url}/@@elastic-controlpanel*
4. In the settings form's *Indexes for which all searches are done through ElasticSearch* section add your all indexes those you mentioned into catalog.xml file, also add *portal_type*
5. Now save and again *Convert Catalog*.

CHAPTER 16

Installation

Install plone.app.fhirfield by adding it to your buildout:

```
[buildout]

...

eggs =
    plone.app.fhirfield [elasticsearch]
```

and then running `bin/buildout`. Go to plone control and install `plone.app.fhirfield` or If you are creating an addon that depends on this product, you may add `<dependency>profile-plone.app.fhirfield:default</dependency>` in `metadata.xml` at profiles.

16.1 configuration

This product provides three plone registry based records `fhirfield.es.index.mapping.nested_fields.limit`, `fhirfield.es.index.mapping.depth.limit`, `fhirfield.es.index.mapping.total_fields.limit`. Those are related to ElasticSearch index mapping setup, if you aware about it, then you have option to modify from plone control panel (Registry).

CHAPTER 17

Links

Code repository:

https://github.com/nazrulworld/plone.app.fhirfield

Continuous Integration:

https://travis-ci.org/nazrulworld/plone.app.fhirfield

Issue Tracker:

https://github.com/nazrulworld/plone.app.fhirfield/issues

set max_map_count value (Linux)

```
` sudo sysctl -w vm.max_map_count=262144 `
```

CHAPTER 18

License

The project is licensed under the GPLv2.

CHAPTER 19

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

plone.app.fhirfield.api, 5

Index

P

plone.app.fhirfield.api (module), 5