Custom Trait: User group

Magnolia Enterprise access required

You need to have the required credentials for using Magnolia Enterprise Pro. You can find suitable settings.xml file for maven and a valid license attached to this page.

Trait

A trait is an attribute of the visitor or visit that you can detect and assign a value to.

In order to create, use and test a new trait you need to define the following elements:

- A **filter** for **detecting** the **trait** on every request.
- A **POJO** to represent the trait itself and **store** all its **attributes**.
- A **rule field** to **store** the **trait values** for a given variant.
- A **voter** for **comparing** rule **field values** stored in jcr against **detected values** in the request.
- A **value field** to **test trait values** and get matching variants.
- A **parameter converter** to **translate** request **parameter values** into **trait objects** and the other way around.

We will learn each of this elements creating an example for user groups trait. The use case is to be able to create personalised content based on the group of the user logged in, so you can have different content for different types of users.

We will be using a module called **personalization-user-custom-trait** for registering java classes and JCR configs. The project with all code needed for this workshop can be found here: https://git.magnolia-cms.com/users/ebguilbert/repos/personalization-user-custom-trait. Please have a look at the module descriptor and the pom to define the needed dependencies in your own module.

1. Start with a web parent project (if you have it already, continue to next step):

```
mvn org.apache.maven.plugins:maven-archetype-plugin:2.4:generate -DarchetypeCatalog=https://nexus.
magnolia-cms.com/content/groups/public/


Choose archetype:
1: https://nexus.magnolia-cms.com/content/groups/public/ -> info.magnolia.maven.archetypes:magnolia-
theme-archetype (An archetype to create STK Theme modules)
2: https://nexus.magnolia-cms.com/content/groups/public/ -> info.magnolia.maven.archetypes:magnolia-
project-archetype (An archetype to create a Magnolia project (a parent pom and a webapp))
3: https://nexus.magnolia-cms.com/content/groups/public/ -> info.magnolia.maven.archetypes:magnolia-
module-archetype (An archetype to create basic Magnolia modules)
4: https://nexus.magnolia-cms.com/content/groups/public/ -> info.magnolia.maven.archetypes:magnolia-
forge-module-archetype (An archetype to create a Magnolia module to be hosted on the Magnolia Forge)
5: https://nexus.magnolia-cms.com/content/groups/public/ -> info.magnolia.maven.archetypes:magnolia-
blossom-module-archetype (An archetype to create Magnolia modules using Blossom)
Choose a number or apply filter (format: [groupId:]artifactId, case sensitive contains): : 2
```

   Modify your POM dependencies to use the enterprise version of Magnolia:

   Parent POM:

```
  <dependency>
        <groupId>info.magnolia.eebundle</groupId>
        <artifactId>magnolia-enterprise-bundle-parent</artifactId>
        <version>${magnoliaVersion}</version>
        <type>pom</type>
        <scope>import</scope>
  </dependency>
```

   Webapp POM:

```
<dependency>
  <groupId>info.magnolia.eebundle</groupId>
  <artifactId>magnolia-enterprise-pro-demo-webapp</artifactId>
  <type>pom</type>
  <version>${magnoliaVersion}</version>
</dependency>
<dependency>
  <groupId>info.magnolia.eebundle</groupId>
  <artifactId>magnolia-enterprise-pro-demo-webapp</artifactId>
  <type>war</type>
  <version>${magnoliaVersion}</version>
</dependency>
```

2. Go to your project folder and create a new module called **personalization-user-custom-trait:**

```
mvn org.apache.maven.plugins:maven-archetype-plugin:2.4:generate -DarchetypeCatalog=https://nexus.
magnolia-cms.com/content/groups/public/


Choose archetype:
1: https://nexus.magnolia-cms.com/content/groups/public/ -> info.magnolia.maven.archetypes:magnolia-
theme-archetype (An archetype to create STK Theme modules)
2: https://nexus.magnolia-cms.com/content/groups/public/ -> info.magnolia.maven.archetypes:magnolia-
project-archetype (An archetype to create a Magnolia project (a parent pom and a webapp))
3: https://nexus.magnolia-cms.com/content/groups/public/ -> info.magnolia.maven.archetypes:magnolia-
module-archetype (An archetype to create basic Magnolia modules)
4: https://nexus.magnolia-cms.com/content/groups/public/ -> info.magnolia.maven.archetypes:magnolia-
forge-module-archetype (An archetype to create a Magnolia module to be hosted on the Magnolia Forge)
5: https://nexus.magnolia-cms.com/content/groups/public/ -> info.magnolia.maven.archetypes:magnolia-
blossom-module-archetype (An archetype to create Magnolia modules using Blossom)
Choose a number or apply filter (format: [groupId:]artifactId, case sensitive contains): : 3
```

3. Add the personalisation dependencies to your module's POM. You will need them in order to import personalisation classes:

```
<dependency>
        <groupId>info.magnolia.personalization</groupId>
        <artifactId>magnolia-personalization-traits</artifactId>
</dependency>
<dependency>
        <groupId>info.magnolia.personalization</groupId>
        <artifactId>magnolia-personalization-preview-app</artifactId>
</dependency>
```

4. Include the new module in your webapp's POM:

```
<dependency>
        <groupId>info.magnolia.personalization.user-custom-trait</groupId>
        <artifactId>personalization-user-custom-trait</artifactId>
        <version>1.0-SNAPSHOT</version>
</dependency>
```

Trait class

First we need to define what the trait would represent. In this case we just want to store a user group name, so it will be class with only one string property:

info.magnolia.personalization.userCustomTrait.trait.UserGroup

```
/**
 * Simple POJO for an user group.
 */
public class UserGroup {

    public static final String EMPTY_VALUE = "empty";

    private String group;

    public UserGroup() {
    }

    public UserGroup(String group) {
        this.group = group;
    }

    public String getGroup() {
                return group;
        }

        public void setGroup(String group) {
                this.group = group;
        }
}
```

To register a trait in Magnolia, it has to be defined inside a folder called traits. Lets define it in our module:

Node name	Value
personalization-user-custom-trait	
traits	
userGroup	
traitClass	info.magnolia.personalization.userCustomTrait.trait.UserGroup

Notice we are using module configuration to register a trait.

Filter

In order to detect the user group of the user making the request, we need a filter. This filter will get the user group from the user in session and will return a trait object based on this group name:

- The constructor receives a trait collector which will store the trait detected.
- The detect method has the detection logic.
- The getTraitClass method returns the trait class.
- There are getters and setters for any configurable property of the filter.

info.magnolia.personalization.userCustomTrait.trait.UserGroupDetectorFilter

```
/**
 * {@link UserGroup} trait filter that detects the user group based on current user.
 *
 * @see info.magnolia.personalization.trait.AbstractTraitDetectorFilter
 */
public class UserGroupDetectorFilter extends AbstractTraitDetectorFilter<UserGroup> {

    protected Map<String, Object> groups = new HashMap<String, Object>();

    @Inject
    public UserGroupDetectorFilter(Provider<TraitCollector> traitCollectorProvider) {
        super(traitCollectorProvider);
    }

    @Override
    public UserGroup detect(HttpServletRequest request, HttpServletResponse response) throws
TraitDetectionException {

        User user = MgnlContext.getUser();
            for (Map.Entry<String, Object> entry : groups.entrySet()){
                    String value = (String)entry.getValue();
                    if(user.inGroup(value)) {
                            return new UserGroup(value);
                    }
            }
        return new UserGroup(UserGroup.EMPTY_VALUE);
    }

    @Override
    protected Class<UserGroup> getTraitClass() {
        return UserGroup.class;
    }

        public Map<String, Object> getGroups() {
                return groups;
        }

        public void setGroups(Map<String, Object> groups) {
                this.groups = groups;
        }

}
```

Notice not all user groups are being detected. This is because the filter only will detect the groups configured in its definition. See below...

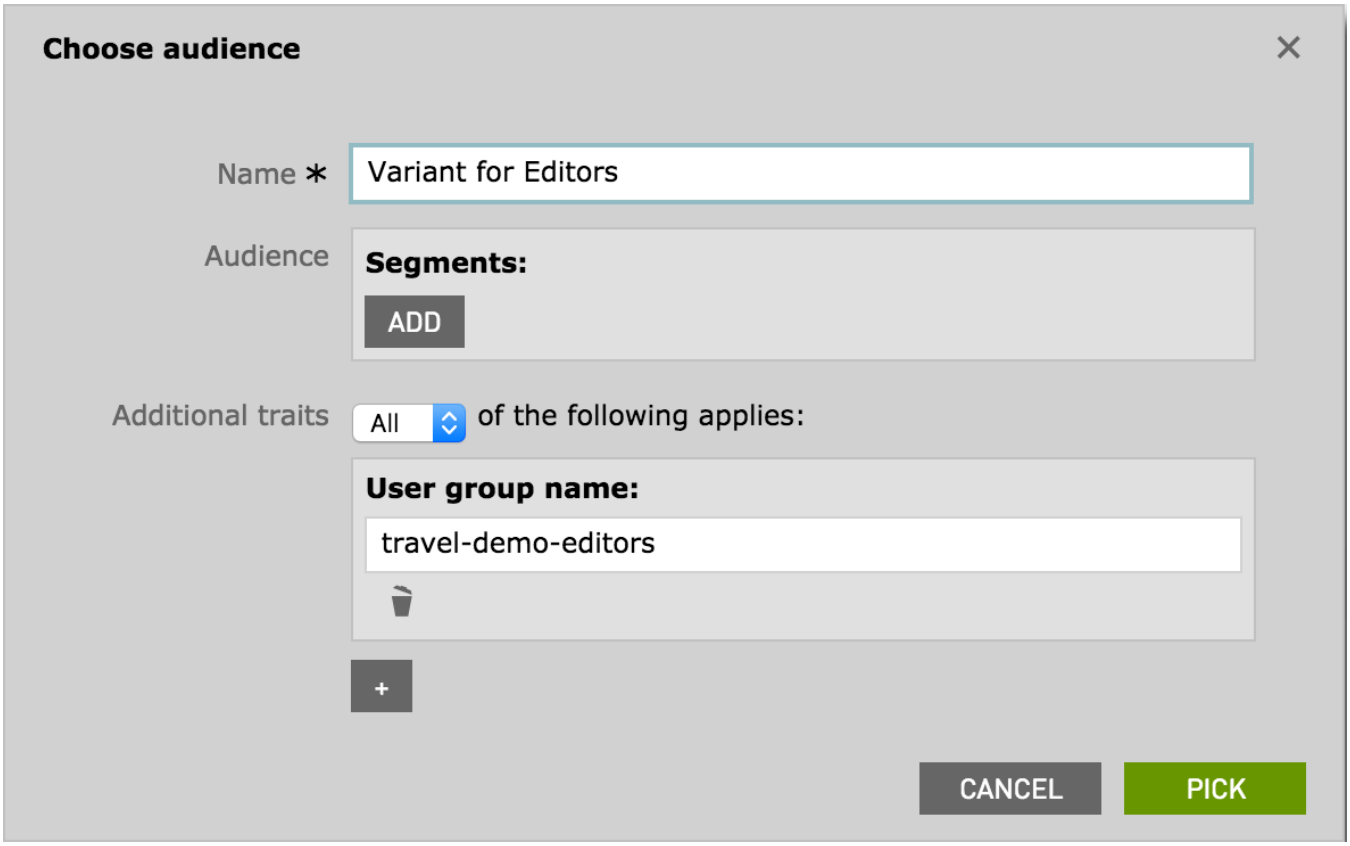
After creating the class you will need to register it as an active filter in the Magnolia filter chain:

Node name	Value
server	
filters	
preview	
userGroup	
groups	

editors	travel-demo-editors
publishers	travel-demo- publishers
class	info.magnolia.personalization.userCustomTrait.trait.UserGroupDetectorFilter
enabled	false

Rule field

A rule field defines values for the trait when you choose an audience. The value stored here will be compared against the value detected by the filter.



Lets define the rule field in our module:

- The trait will be using a text field. You can define any field available in Magnolia.
- You can define transformers and converters as with any other field in Magnolia.

Node name	Value
personalization-user-custom-trait	
traits	
userGroup	

ruleField	
class	info.magnolia.ui.form.field.definition.TextFieldDefinition
type	String
traitClass	info.magnolia.personalization.userCustomTrait.trait.UserGroup

Notice we created a traits folder in order to register our trait called userGroup.

Voter

The voter decides whether the visitor or visit matches the rule or not. When the rule is met, personalised content is delivered.

The voter receives a trait collector which holds the traits detected, and also receives the rule field with the value configured in the audience. The voter's logic is to compare 2 trait values:

- The trait detected: This is being hold by the trait collector.
- The trait configured as a rule for an audience. This is being hold by the rule field.

Lets have a look at our voter for the user group trait:

- The rule field is a property of the class with a getter and setter.
- The bool method has the comparison logic.

info.magnolia.personalization.userCustomTrait.trait.UserGroupVoter

```
/**
 * Voter for {@link UserGroup}.
 */
public class UserGroupVoter extends AbstractBoolVoter<TraitCollector> {

    private String ruleField;

    @Override
    protected boolean boolVote(final TraitCollector traitCollector) {
        if (traitCollector != null) {
            final UserGroup country = traitCollector.getTrait(UserGroup.class);
            if (country != null) {
                return StringUtils.equalsIgnoreCase(country.getGroup(), getRuleField());
            }
        }
        return false;
    }

    public String getRuleField() {
                return ruleField;
        }

        public void setRuleField(String ruleField) {
                this.ruleField = ruleField;
        }

        @Override
    public String toString() {
        return "userGroup=" + ruleField;
    }

}
```

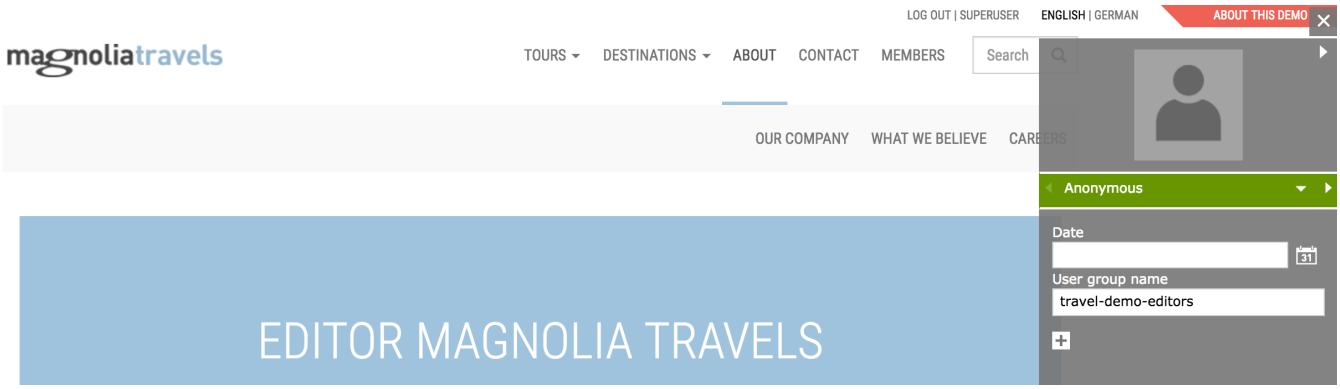
The voter needs to be registered as a property of the trait in our module:

Node name	Value
personalization-user-custom-trait	
traits	
userGroup	
ruleField	
traitClass	info.magnolia.personalization.userCustomTrait.trait.UserGroup
voterClass	info.magnolia.personalization.userCustomTrait.trait.UserGroupVoter

Notice we created a traits folder in order to register our trait called userGroup.

Value field

A value field defines values for the trait when you are using the Preview App. The value stored here will be compared against the values stored as rules.



Lets define the value field in our module:

- The trait will be using a text field. You can define any field available in Magnolia.
- You can define transformers and converters as with any other field in Magnolia.

Node name	Value
personalization-user-custom-trait	
traits	
userGroup	
ruleField	
valueField	
class	info.magnolia.ui.form.field.definition.TextFieldDefinition
type	String
traitClass	info.magnolia.personalization.userCustomTrait.trait.UserGroup
voterClass	info.magnolia.personalization.userCustomTrait.trait.UserGroupVoter

Notice we created a traits folder in order to register our trait called userGroup.

Preview parameter converter

The preview parameter converter is used internally by the Preview App to translate the trait values filled in the app into trait objects in the trait collector so the voter can work as intended.

For the user group trait, the converter looks like this:

- The fromString method translates a string value into a trait object that can be handled by the voter.
- The toString method translates a trait object into a string than can be handled by the Preview App.

info.magnolia.personalization.userCustomTrait.trait.UserGroupParameterConverter

```
/**
 * Parameter converter for {@link UserGroup}.
 */
public class UserGroupParameterConverter implements PreviewParameterConverter<Object> {

           @Override
            public UserGroup fromString(String string) {
                return new UserGroup(string);
            }

            @Override
            public String toString(Object object) {
                if (object instanceof String) {
                    if (UserGroup.EMPTY_VALUE.equals(object)) {
                        return null;
                    }
                    return (String) object;
                }
                if (object instanceof UserGroup) {
                    return ((UserGroup) object).getGroup();
                }
                return null;
            }
}
```

The converter needs to be registered as a property of the trait configuration:

Node name	Value
personalization-user-custom-trait	
traits	
userGroup	
ruleField	
valueField	
converterClass	info.magnolia.personalization.userCustomTrait.trait.UserGroupParameterConverter
traitClass	info.magnolia.personalization.userCustomTrait.trait.UserGroup
voterClass	info.magnolia.personalization.userCustomTrait.trait.UserGroupVoter

Trait Config

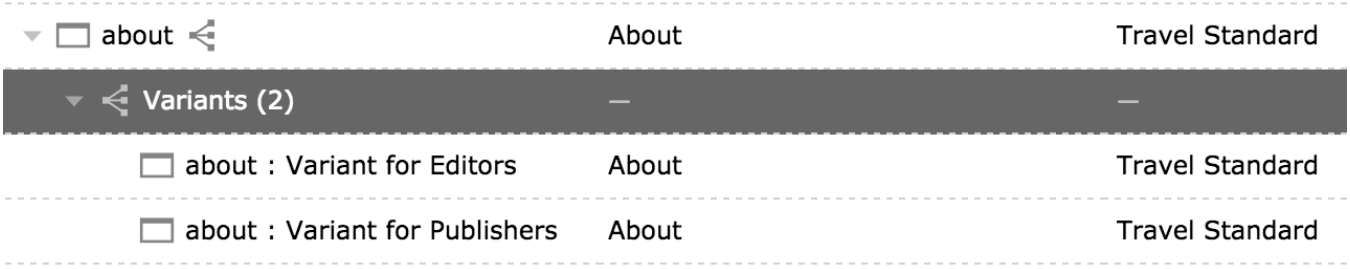
The complete configuration for the trait:

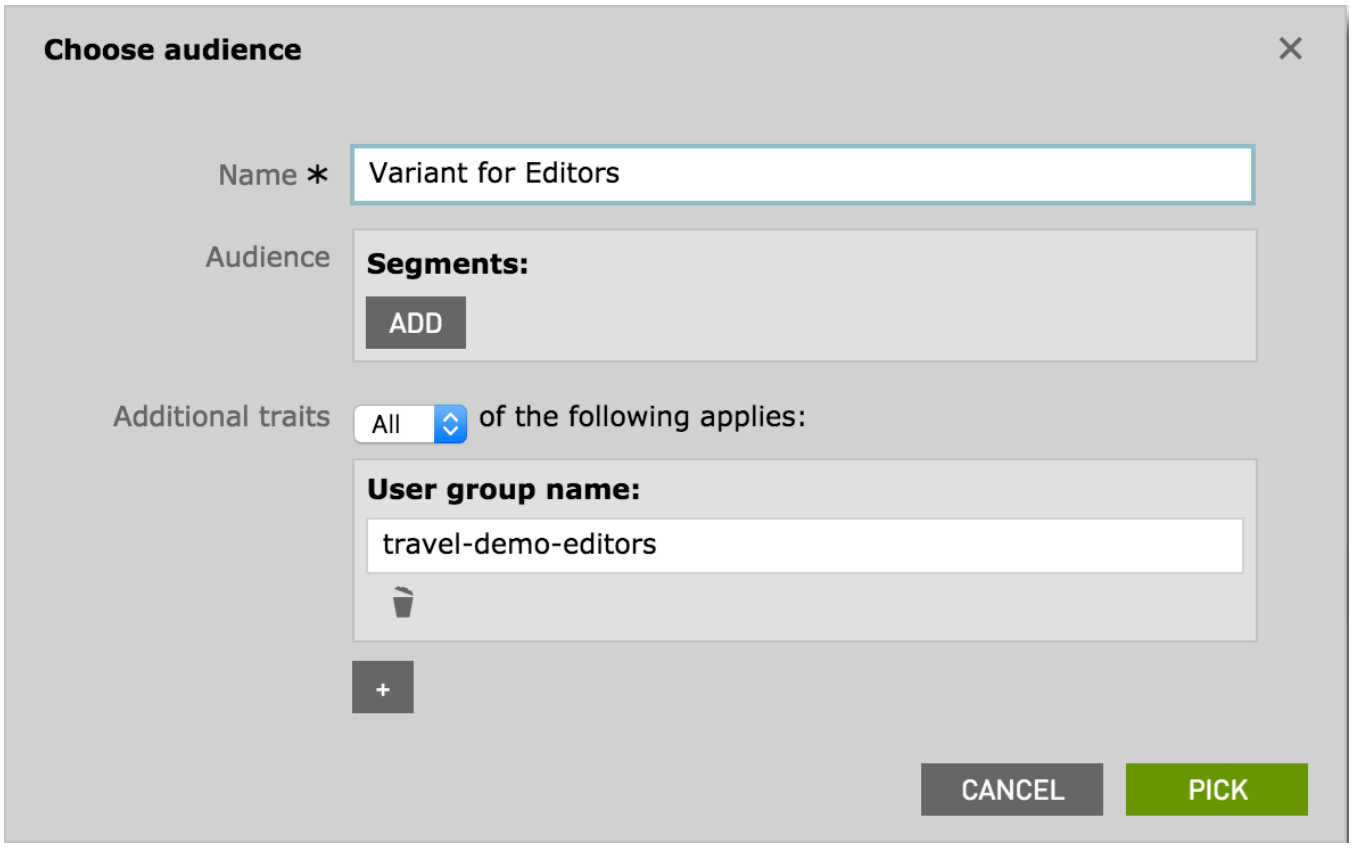
Node name	Value
personalization-user-custom-trait	
traits	

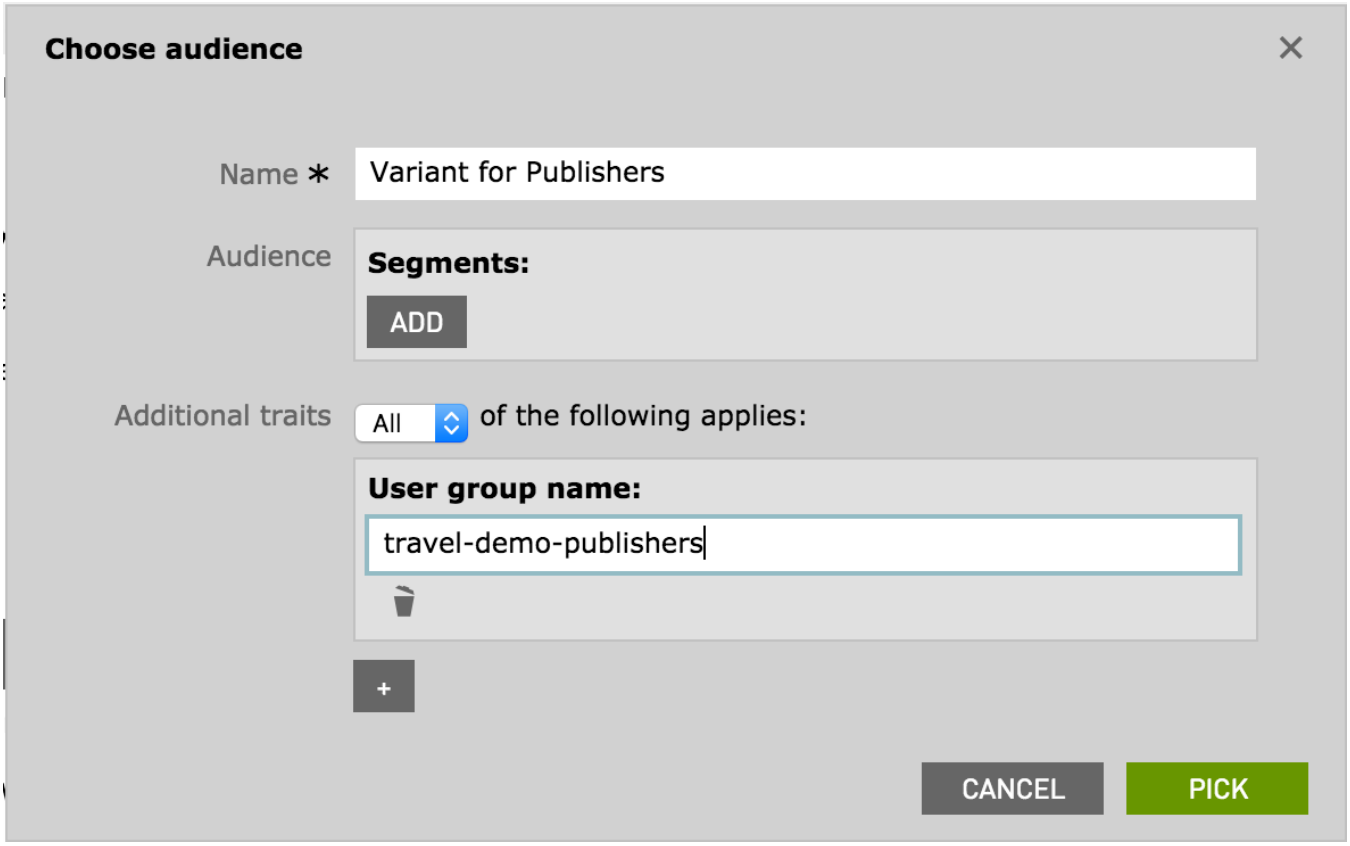
userGroup	
ruleField	
class	info.magnolia.ui.form.field.definition.TextFieldDefinition
type	String
valueField	
class	info.magnolia.ui.form.field.definition.TextFieldDefinition
type	String
converterClass	info.magnolia.personalization.userCustomTrait.trait.UserGroupParameterConverter
defaultPreviewTrait	true
defaultRuleTrait	true
traitClass	info.magnolia.personalization.userCustomTrait.trait.UserGroup
voterClass	info.magnolia.personalization.userCustomTrait.trait.UserGroupVoter

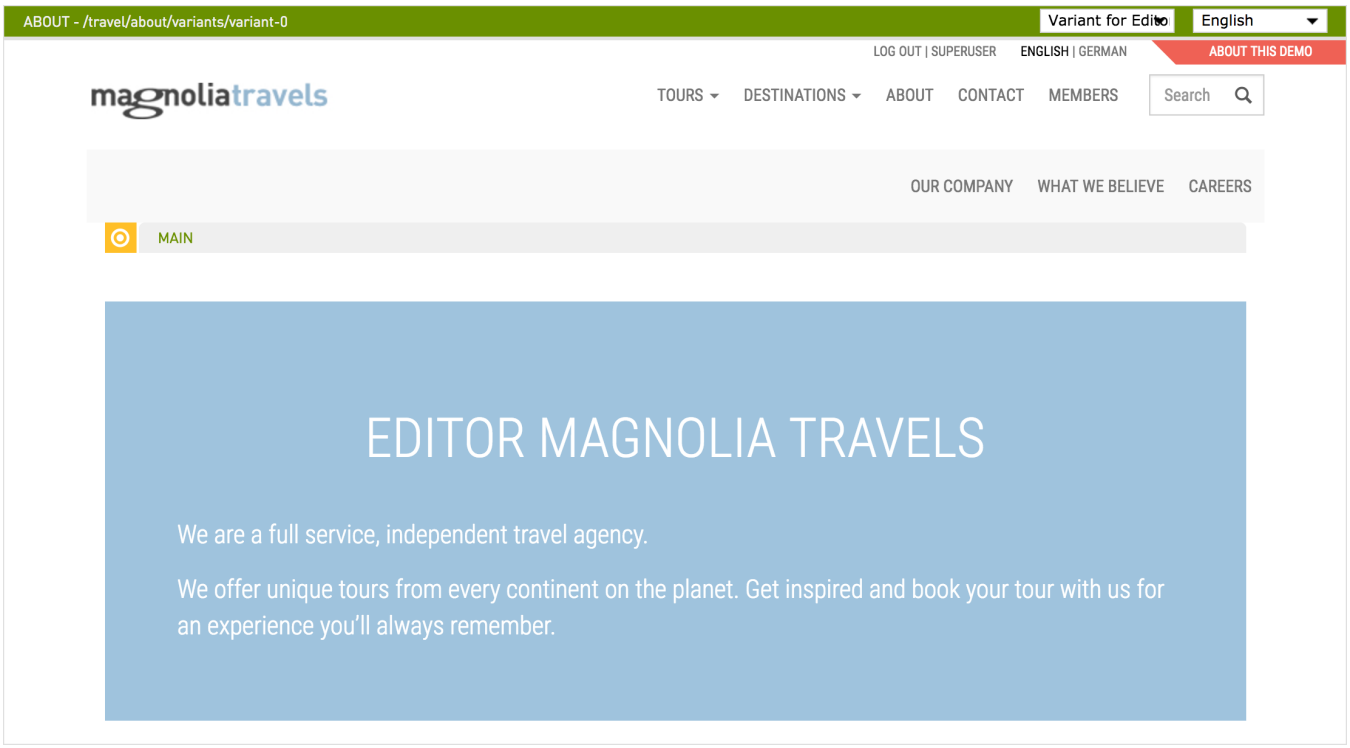
Notice the default properties to add the rule and value fields by default in choose audience dialogs and Preview App.

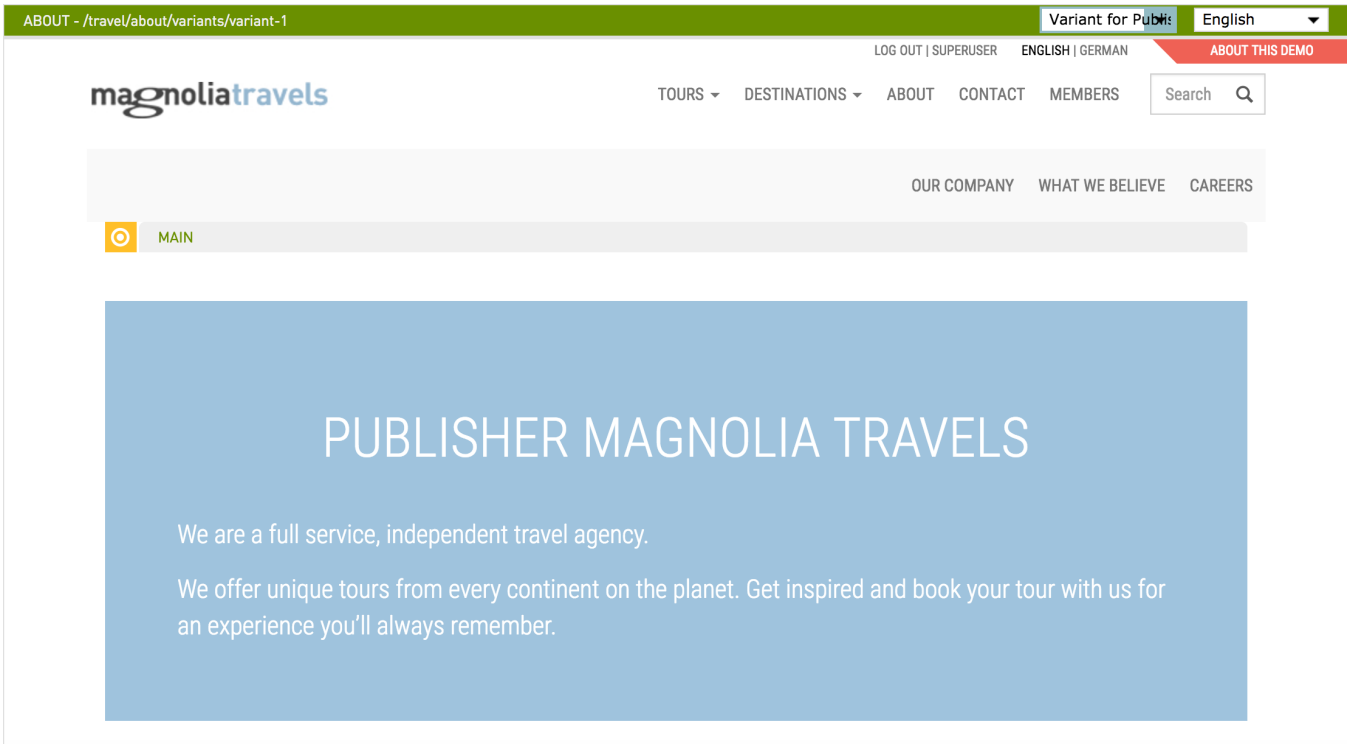
Test the trait with variants




Choose audience
Name *
Variant for Editors
Audience
Segments:
ADD
Additional traits
All
of the following applies:
User group name:
travel-demo-editors
+
CANCEL
PICK


Choose audience
Name *
Variant for Publishers
Audience
Segments:
ADD
Additional traits
All
of the following applies:
User group name:
travel-demo-publishers
+
CANCEL
PICK





« More examples Exercise: Device trait »