User Guide 0.x

Table of Contents

MetaConstraints

inPlace

This property is used to configure the type of relation (one-to-many, many-to-one, one-to-one) that will be displayed.

A relation "inPlace:true" hasn't external CRUD, the CRUD is performed within the editView that contains it.
A relation "inPlace:false" has external CRUD, the CRUD is within the editView of the related DomainClass.

usage

```
constraint =
{
    .....
    someDomainProperty(inPlace:[true|false])
    .....
}
```

createView

This property is used when we want a component is visible or not visible in creationtime

If createView is seted true, the component will visible in creationtime, if is false not

usage

```
constraint =
{
    .....

someDomainProperty(createView:[true|false])
    .....
}
```

editView

Like createView but edittime

usage

```
constraint =
{
    .....
    someDomainProperty(editView:[true|false])
    .....
}
```

defaultValue

This property is used when we want set a default value for a property.
It can be combined with the properties createView and editView to set the default value of the property when their control is not visible.

usage

```
constraint =
{
    .....
    someDomainProperty(defaultValue:"aValue")
    .....
}
```

Validations

GFS generates the necessary code to validate in Frontend some constraints, so that it is not necessary to fulfil a request to Backend.

Constraints supported are:

- blank
- url
- email
- size

- minSize
- maxSize
- min
- max
- range

If you want to know more about its use, you can read the Grails' Official Doc

The constraints that are not yet supported by GFS and are supported by Grails, we used the error messages sent from the Backend. These error messages are i18n support.

Controls

GFS has available some visual controls that offer to the user different ways of representing the attributes of the DomainClass on the "Flex View".

inList

ComboBox - Default Value
AutoComplete

usage

```
constraint =
{
    .....
    someDomainProperty(inList:[list of
items],widget:'autocomplete')
    .....
}
```

Numbers

NumericStepper - Default Value
TextInput
VSlider
HSlider

usage

```
constraint =
{
    .....

someNumberDomainProperty(widget:'textinput')

someNumberDomainProperty(widget:'vslider')

someNumberDomainProperty(widget:'hslider')
    .....
}
```

Strings

TextInput - Default Value
TextArea
RichText
ColorPicker
FileUpload upload file
ImageUpload upload file widget image
SnapshotUpload upload image from webcam
Password two text input with password format
usage

```
constraint =
{
    .....

someStringDomainProperty(widget:'textarea')

someStringDomainProperty(widget:'richtext')

someStringDomainProperty(widget:'colorpicker
')

someStringDomainProperty(widget:'fileupload'
)

someStringDomainProperty(widget:'imageupload
')

someStringDomainProperty(widget:'snapshotupl
oad')

someStringDomainProperty(widget:'password')
    .....
}
```

Date

CBKDateField - the only control for dates at the moment

inPlace

one-to-one inPlace:true - Custom Componet that show relation information (at the moment not supported inPlace:false)

many-to-one inPlace:false - ComboBox (at the moment not supported inPlace:true)

one-to-may inPlace:true - DataGrid (at the moment not supported inPlace:false)

i18n

Since version 0.2 GFS generated applications have multi-language support. This means that for a domain class GFS will generate all the messages for the languages set in the properties file.
usage

Properties file

grails-app/i18n/i18n.properties has all the properties necessary to set i18n support. This properties are:
*locale.default=en identifies the application default language
*locale.locales=de, es, fr, pt_BR represents all languages that will be supported by the application.

Script

generate-i18n is responsable for all the messages generated and it can be used in two ways:
$grails generate-i18n "*"
.properties files are generated for all the languages set in i18n.properties for each domain class
$grails generate-18n aDomainClass
.properties files are generated for all the languages set in i18n.properties for the domain class

Useful tip

When you execute "$generate-all-flex aDomainClass" only the properties for the default language are generated. If you need an application with support for more than one language, this should be the correct way to do it:

```
$grails generate-all-flex "*"
$grails generate-i18n "*"
$grails flex-tasks
$grails run-app
```

MetaConstraint

i18n
This property allows you to define the local for an attribute on your domain class.
If it's setted, GFS replace key of message_locale.properties. Otherwise GFS tranlates via google translator.

usage

```
constraint =
{
  .....
  someDomainProperty(i18n:[es:"Propiedad de
Dominio",en:"Some Domain Property"])
  .....
}
```

Dynamic Jasper Integration

Since version 0.2, GFS generates the UI necessary to obtain dynamic reports using DJ Report. The UI has an accordion and a JPG display (GFS transforms the PDF file into a JPG image). The accordion holds all the filters needed to perform the query and the columns setup.

The PDF gets displayed inside an iframe. Adobe PDF reader is required for the application to display the generated report.

usage

```
//Just add "reportable" as a property for
your domain class.

class aClass
{
  static def reportable = [:]
}
```

Annotations

FlexScaffoldProperty

- **labelField** : The label field is displayed in edit-view for the external relation. (Old way to managing relation)
- **generate** : Used to define if GFS should generate CRUD operations or not.

File Upload

Since version 0.2 GFS can upload files. see string widget for it's configuration

If you need to save files into specific folder, you must be set filepath property in Config.groovy

usage

```
environments {
    production {
        grails.serverURL =
"http://www.changeme.com"
        filepath = "/www/files"
    }
    development {
        filepath =
"/home/user/projects/gfs/files"
    }
}
```

Spring-Security

Since v0.2.2 GFS solves security integration through stark-security (User & Role into DB or LDAP) plugin

The required steps to configure it are:

> ⚠️ We assume that gfs is installed into the target application

```
gfs@aProject $ grails install
stark-security-install-full
```

Edit grails-app/conf/StarkSecurityConfig.config and modify as below

```
authorizations = [
   '/': Role.ALL_ROLES,
   ...
   ...
 ]

 to

 authorizations = [
  '/**': Role.ALL_ROLES,
  ...
  ...
 ]
```

Add to grails-app/conf/BootStrap.groovy as below

```
import
org.codehaus.groovy.grails.plugins.starksecu
rity.PasswordEncoder

class BootStrap {

    def init = { servletContext ->

 if (!User.findByUsername("admin")) {

          def superUser = new User(username:
'admin',
                                   password:
PasswordEncoder.encode('admin', 'SHA-256',
true))

  def role = new
Role(authority:"ROLE_USER",description:"Role
users")

  superUser.addToRoles(role)

          superUser.save()
  }

   }

   def destroy = {
   }
}
```

Add to grails-app/conf/Config.groovy the following line

```
gfs.security = true
```

⚠️ If you want generete User and Role CRUD take a look to this tutorial

⚠️ If you used FlexBuilder you must set-define=GFS::security,false to true (“-define=GFS::security,true”) in the Flex Compiler options (flex arguments), this property will show login dialog if is setted to true. By default it’s false and in command-line compiling (grails flex-tasks) this property will be retrieved from Config.groovy (gfs.security = [true|false]).

Tab Groupping

Since v0.2.2 GFS supports tab gropping for domain classes. This means that several domain classes (tabs) grouped in one super tab (Thank you **Nicolas Domina**).

To use this feature you may set a groupName into domain classes as below

```
class Foo {
 String nameFoo
 String lastNameFoo

 static def groupName = "ADM Foo"
}

class Bar {
 String nameBar
 String lastNameBar

 static def groupName = "ADM Foo"
}

//GFS generetes a Super Tab on upper tab
navigator named "ADM Foo" that contains Foo
tab and Bar tab at the bottom
```

Actions

Since v0.2.2 GFS supports actions

DataGrid

It supports two features. multiselection and actions buttons.

Multiselection: allow select one or more items on the datagrid (like delete multirows)

Actions buttons: show an action button in each row so you can add bussines logic into method at [DomainClass]Service

Example

```
class Foo {
 String name

 static action =
[datagrid:[multiselection:true,actions:["act
ion1"]]]
}
//You could declare as many actions buttons
as you need eg:
actions:["action1","action2"]
```

Scripts

create-cairngorm - Creates Cairngorm event and command classes for the given base name
$ grails create-cairngorm aCommandAndEventName

flex-tasks - Compiles Flex sources
$ grails flex-tasks

generate-all-flex - Generates all Flex's artifacts for the given domain class
$ grails generate-all-flex aDomainClass

$ grails generate-all-flex "*" see Annotations "generate"

generate-command - Generates Cairngorm's commands for the given domain class (to perform the CRUD)
$ grails generate-command aDomainClass

generate-delegate - Generates business delegate for the given domain class
$ grails generate-delegate aDomainClass

generate-event - Generaes Cairngorm's events for the given domian class (to perform the CRUD)
$ grails generate-event aDomainClass

generate-i18n - Generates .properties of each locale for the given domain class
$ grails generate-i18n aDomainClass
$ grails generate-i18n "*"

generate-model - Generates DomainModel for the given domain class (to perform the CRUD)
$ grails generate-model aDomainClass

generate-service - Generates a Grails Service CRUD class for the given domain class
$ grails generate-service aDomainClass

generate-view - Generates MXML views for the given domain class

$ grails generate-view aDomainClass

generate-vo - Generates ActionScript VO class for the given domain class
$ grails generate-vo aDomainClass