
GRAILS


spring
source

Flex Plugin - Reference Documentation

Authors: Burt Beckwith

Version: 0.4

Table of Contents

1 Introduction to the Flex Plugin

The Flex plugin makes it easier to build Grails-powered Rich Internet Applications using Adobe Flex as the front-end client. It depends on the Blaze DS plugin which configures Adobe BlazeDS to provide remoting and messaging as well as the Spring Security Core plugin to secure your application.
In development mode you edit MXML and ActionScript files which get compiled into .swf files and support reloading without needing to restart (just like GSPs). When building a war for production deployment your MXML files get precompiled for performance.

Plugin version history

0.2 (March 6, 2008)

- Upgraded to BlazeDS release
- Destination name is now configurable (GRAILSPLUGINS-226)

0.4 (TODO)

- Completely rewritten
- Depends on the BlazeDS plugin
- Depends on the Spring Security Core plugin to provide support for Spring Security 3
- Uses Spring Flex version 1.5

1.1 Getting Started

The first step is installing the plugin:

```
grails install-plugin flex
```

This will transtitively install the BlazeDS, Spring Security Core, and Spring Security ACL plugins. Refer to the documentation for these plugins for configuration options. At a minimum you'll need to run the `s2-quickstart` script to configure Spring Security Core, e.g.

```
grails s2-quickstart com.yourcompany.yourapp User Role
```

The BlazeDS and Spring Security ACL plugins have no required initialization steps.

The next step is configuring the Flex plugin:
You either need to have a `FLEX_HOME` environment variable set, or set the `grails.plugin.flex.home` property in `grails-app/conf/Config.groovy`. This must point at the location of a Flex SDK, e.g.

```
grails.plugin.flex.home = 'c:/devtools/flex'
```

Once that's configured, run the flex-init script to copy the required development files to your project (in the `web-app/WEB-INF/flex` folder):

```
grails flex-init
```

2 Configuration

There are several configuration options for the plugin, but most are related to mxml compilation and html wrapper file generation.

Core properties
There are only two 'core' properties and only one is required - `grails.plugin.flex.home` - and only if the `FLEX_HOME` environment variable is not set.

Property	Default	Meaning
grails.plugin.flex.home	none, must be set	Location of your Flex SDK
grails.plugin.flex.webtier.compiler.enabled	`true` in development	whether the webtier MXML compiler is enabled

MXMLC properties
There are also several properties that configure mxml compilation, both automatic precompilation when building a war file and explicit compilation when running the compile-mxml script.

Property	Default	Meaning
grails.plugin.flex.precompileMxml.enabled	`false`	Set to `true` to precompile mxml files to swf files when creating a war
grails.plugin.flex.precompileMxml.files	none	A list of file names (relative to the `web-app` folder) to precompile
grails.plugin.flex.mxmlc.keepGenerated	`false`	Set to `true` keep the files generated by the compiler
grails.plugin.flex.mxmlc.extraSourcePaths	none	A list of extra source paths
grails.plugin.flex.mxmlc.extraLibPaths	none	A list of extra library paths
grails.plugin.flex.mxmlc.compileTimeConstants	none	A map of compile-time constants
grails.plugin.flex.mxmlc.contextRoot	application name	The context root of the web application, used to calculate AMF channel URLs
grails.plugin.flex.mxmlc.incremental	`false`	When incremental compilation is enabled, the compiler inspects changes to the bytecode between revisions and only recompiles the section of bytecode that has changed.
grails.plugin.flex.mxmlc.debug	`false`	If `true` generates a debug SWF file which includes line numbers and filenames of all source files.
grails.plugin.flex.mxmlc.verboseStacktraces	`false`	If `true` generates source code that includes line numbers for display in stacktraces when a runtime error occurs.
grails.plugin.flex.mxmlc.actionscriptFileEncoding	none	Sets the .as file encoding so that the compiler correctly interprets ActionScript files.
grails.plugin.flex.mxmlc.accessible	`false`	Enables accessibility features
grails.plugin.flex.mxmlc.headless	`false`	Enables the headless implementation of the Flex compiler; required to use fonts and SVG on UNIX systems without X Windows.
grails.plugin.flex.mxmlc.optimize	`true`	Enables the ActionScript optimizer which reduces file size and increases performance by optimizing the SWF file's bytecode.

HTML wrapper properties

The remaining properties configure the generation of an html wrapper to load a swf file.

Property	Default	Meaning
grails.plugin.flex.mxmlc.htmlWrapper.create	`false`	if `true` then an html wrapper file will be created when precompiling
grails.plugin.flex.mxmlc.htmlWrapper.title	MXML file name	The HTML title tag value
grails.plugin.flex.mxmlc.htmlWrapper.extension	gsp	The extension for the html file
grails.plugin.flex.mxmlc.htmlWrapper.history	`false`	If true creates resources for history tracking
grails.plugin.flex.mxmlc.htmlWrapper.height	`100%`	The height of the swf in the page
grails.plugin.flex.mxmlc.htmlWrapper.width	`100%`	The width of the swf in the page
grails.plugin.flex.mxmlc.htmlWrapper.bgcolor	`#ffffff`	The background color for the page
grails.plugin.flex.mxmlc.htmlWrapper.version.major	`10`	The minimum required Flash major version
grails.plugin.flex.mxmlc.htmlWrapper.version.minor	`0`	The minimum required Flash minor version
grails.plugin.flex.mxmlc.htmlWrapper.version.revision	`0`	The minimum required Flash revision version

3 General Usage

Runtime configuration
The plugin (with help from the BlazeDS plugin) manages the configuration of BlazeDS and Flex for you. This includes configuring the webtier mxml compiler and support (`flex.webtier.server.j2ee.MxmlServlet` and `flex.webtier.server.j2ee.SwfServlet`) if enabled (true by default in development mode) and the `flex.webtier.server.j2ee.ForbiddenServlet` which denies access to source code files.
When you run the flex-init script, initial `flex-config.xml` and `flex-webtier-config.xml` configuration files are generated in the `/WEB-INF/flex/` folder, and required resources from your Flex SDK are also copied there. You can customize all of these but the initial configuration should be sufficient to get started.

Development mode
In development mode you can create and edit .mxml and .as files in the web-app folder or subfolders and load the .mxml files in a web browser. They'll get compiled into a .swf file and loaded if it's the first time you've loaded them, or if one of the source files (the top-level .mxml file or any referenced .mxml or .as file) has changed.

Production mode
In production mode files are precompiled (like GSPs) so there's no initial compilation lag and performance is maximized. Precompilation is configurable via settings and you can also explicitly compile the files yourself, either with the Scripts script or by using the SDK tools directly.

Grails services as remote destinations
As described in the BlazeDS plugin documentation, it's simple to access a Grails service as a remote service from Flex - just annotate the service with the `org.springframework.flex.remoting.RemotingDestination` annotation. BlazeDS handles invoking method calls and marshalling parameters and return values for you.
One thing that's important to note is that although there is a `services-config.xml` configuration file (created when the BlazeDS plugin is installed) you probably won't need to make many changes there. In typical Flex applications you would register remote services using XML, but Grails services are well suited as candidates for remote services.

Flash Builder integration
You can easily create Flex-based applications using your IDE or a text editor but using Flash Builder is a lot more convenient. The standard project layout for Flash Builder isn't the same as for Grails though, so you can run the integrate-with-flash-builder script to update your Eclipse/STS .project file and create .flexProperties and .actionScriptProperties files, all configured with settings that are compatible for both environments.

Scripts
You can use the plugins file-generating scripts to help create MXML and ActionScript classes. These include create-mxml which creates a basic MXML file, create-actionscript-class which creates a basic ActionScript class, and generate-actionscript-class which generates an ActionScript class based on one of your domain classes for use with remote services.

Manual configuration
You're not limited to using Grails services as the server-side implementation of remote services. You can use any class like you would in a non-Grails application, and the best place to configure this is in `grails-app/conf/spring/resources.groovy` using the BeanBuilder syntax equivalent of the Spring Flex XML configuration.
For example, this resources.groovy file contains four messaging destinations and two remoting destinations. Remoting destinations are configured like any other Spring bean, with the addition of a `flex.'remoting-destination'()` child element, optionally with configuration options. See the Spring Flex documentation for more information on what's available.

```
import flex.management.jmx.MBeanServerGateway
beans = {
   xmlns flex: 'http://www.springframework.org/schema/flex'
   flex.'message-destination'(id: 'chat')
   flex.'message-destination'(id: 'secured-chat', 'send-security-constraint': 'trusted')
   flex.'message-destination'(id: 'simple-feed')
   flex.'message-destination'(id: 'market-feed', 'allow-subtopics': true, 'subtopic-separato
   securityHelper(Security3Helper) {
      flex.'remoting-destination'()
   }
   RuntimeManagement(MBeanServerGateway) {
      flex.'remoting-destination'(channels: 'my-amf, my-secure-amf')
   }
}
```

4 Tutorial

1. Create your Grails application.

```
$ grails create-app FlexContacts
$ cd FlexContacts
```

2. Install the plugin.

```
$ grails install-plugin flex
```

3. Initialize Spring Security.

```
$ grails s2-quickstart grails.demo.flexcontacts User Role
```

4. Configure the Flex SDK location.
If you don't have a `FLEX_HOME` environment variable set, configure the location of your Flex SDK in `grails-app/conf/Config.groovy`:

```
grails.plugin.flex.home = '/path/to/your/flex/sdk'
```

5. Initialize the Flex plugin.

```
$ grails flex-init
```

6. Create the Contact domain class.

```
$ grails create-domain-class grails.demo.flexcontacts.Contact
```

Update the domain class with this code:

```
package grails.demo.flexcontacts
class Contact implements Serializable {
   static final long serialVersionUID = 1
   String firstName
   String lastName
   String address
   String city
   String state
   String zip
   String phone
   String email
   static constraints = {
      firstName size: 1..50, blank: false
      lastName size: 1..50, blank: false
      address size: 1..50, blank: false
      city size: 1..50, blank: false
      state size: 1..20, blank: false
      zip size: 1..20, nullable: true
      phone size: 1..50, nullable: true
      email size: 1..50, blank: false
   }
   static List<Contact> findByName(String name) {
      executeQuery("from Contact where UPPER(CONCAT(firstName, ' ', lastName)) LIKE :name OR
         [name: "%" + name.toUpperCase() + "%"])
   }
}
```

7. Create the Contact service.

```
$ grails create-service grails.demo.flexcontacts.Contact
```

Update the service with this code:

```
package grails.demo.flexcontacts
import org.springframework.flex.remoting.RemotingDestination
@RemotingDestination(channels = ['my-amf'])
class ContactService {
   List<Contact> findByName(String name) {
      Contact.findByName name
   }
   List<Contact> findAll() {
      Contact.list()
   }
   Contact findById(long id) {
      Contact.get id
   }
   Contact create(Contact contact) {
      contact.id = null
      contact.save()
      contact
   }
   boolean update(Contact contact) {
      Contact fromDb = Contact.get(contact.id)
      if (!fromDb) {
         return false
      }
      fromDb.properties = contact.properties
      fromDb.validate() && fromDb.save()
   }
   boolean remove(Contact contact) {
      Contact.get(contact.id)?.delete()
      true
   }
}
```

8. Create the primary MXML file.

```
$ grails create-mxml web-app/Main.mxml
```

Update the MXML file with this code:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns="*">
   <mx:Script><![CDATA[
      import mx.rpc.events.FaultEvent;
      import mx.controls.Alert;
      import mx.collections.ArrayCollection;
      import mx.rpc.events.ResultEvent;
      [Bindable] private var contacts:ArrayCollection;
      private function resultHandler(event:ResultEvent):void {
         contacts = event.result as ArrayCollection
      }
      private function faultHandler(event:FaultEvent):void {
         Alert.show(event.fault.faultDetail);
      }
      public function openContact(contact:Contact):void {
         var children:Array = tn.getChildren();
         for (var i:int = 0; i<children.length; i++) {
            if (ContactForm(children[i]).contact.id == contact.id) {
               tn.selectedChild = children[i];
               return;
            }
         }
         var form:ContactForm = new ContactForm();
         tn.addChild(form);
         form.contact = contact;
         tn.selectedChild = form;
      }
   ]]></mx:Script>
   <mx:RemoteObject id="ro" destination="contactService" fault="faultHandler(event)">
      <mx:method name="findByName" result="resultHandler(event)"/>
   </mx:RemoteObject>
   <mx:ApplicationControlBar width="100%">
      <mx:TextInput id="searchStr"/>
      <mx:Button label="Search" click="ro.findByName(searchStr.text)"/>
      <mx:Button label="New Contact" click="openContact(new Contact())"/>
   </mx:ApplicationControlBar>
   <mx:HDividedBox width="100%" height="100%">
      <mx:DataGrid id="dg" dataProvider="{contacts}" width="30%" height="100%"
         doubleClickEnabled="true"
         doubleClick="openContact(dg.selectedItem as Contact)">
         <mx:columns>
            <mx:DataGridColumn dataField="firstName" headerText="First Name"/>
            <mx:DataGridColumn dataField="lastName" headerText="Last Name"/>
         </mx:columns>
      </mx:DataGrid>
      <mx:TabNavigator id="tn" width="70%" height="100%"/>
   </mx:HDividedBox>
</mx:Application>
```

9. Create the contact form MXML file.

```
$ grails create-mxml web-app/ContactForm.mxml
```

Update the MXML file with this code:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml" width="100%" height="100%"
   backgroundColor="#FFFFFF"
   label="{contact.id>0?contact.firstName+' '+contact.lastName:'New Contact'}">
   <mx:Script><![CDATA[
      import mx.rpc.events.FaultEvent;
      import mx.rpc.events.ResultEvent;
      import mx.controls.Alert;
      [Bindable] public var contact:Contact;
      private function save():void {
         contact.firstName = firstName.text;
         contact.lastName = lastName.text;
         contact.email = email.text;
         contact.phone = phone.text;
         contact.address = address.text;
         contact.city = city.text;
         contact.state = state.text;
         contact.zip = zip.text;
         if (contact.id == 0) {
            ro.create(contact);
         }
         else {
            ro.update(contact);
         }
      }
      private function create_resultHandler(event:ResultEvent):void {
         contact.id = event.result.id;
      }
      private function deleteItem():void {
         ro.remove(contact);
      }
      private function remove_resultHandler(event:ResultEvent):void {
         parent.removeChild(this);
      }
      private function faultHandler(event:FaultEvent):void {
         Alert.show(event.fault.faultDetail);
      }
   ]]></mx:Script>
   <mx:RemoteObject id="ro" destination="contactService">
      <mx:method name="create" result="create_resultHandler(event)"/>
      <mx:method name="remove" result="remove_resultHandler(event)"/>
   </mx:RemoteObject>
   <mx:Form>
      <mx:FormItem label="Id">
         <mx:TextInput text="{contact.id}" enabled="false"/>
      </mx:FormItem>
      <mx:FormItem label="First Name">
         <mx:TextInput id="firstName" text="{contact.firstName}"/>
      </mx:FormItem>
      <mx:FormItem label="Last Name">
         <mx:TextInput id="lastName" text="{contact.lastName}"/>
      </mx:FormItem>
      <mx:FormItem label="Email">
         <mx:TextInput id="email" text="{contact.email}"/>
      </mx:FormItem>
      <mx:FormItem label="Phone">
         <mx:TextInput id="phone" text="{contact.phone}"/>
      </mx:FormItem>
      <mx:FormItem label="Address">
         <mx:TextInput id="address" text="{contact.address}"/>
      </mx:FormItem>
      <mx:FormItem label="City">
         <mx:TextInput id="city" text="{contact.city}"/>
      </mx:FormItem>
      <mx:FormItem label="State">
         <mx:TextInput id="state" text="{contact.state}"/>
      </mx:FormItem>
      <mx:FormItem label="Zip">
         <mx:TextInput id="zip" text="{contact.zip}"/>
      </mx:FormItem>
   </mx:Form>
   <mx:HBox left="8" bottom="8">
      <mx:Button label="Close" click="parent.removeChild(this)"/>
      <mx:Button label="Save" click="save()"/>
      <mx:Button label="Delete" click="deleteItem()"/>
   </mx:HBox>
</mx:Canvas>
```

10. Generate the Contact ActionScript class.

```
$ grails generate-actionscript-class Contact grails.demo.flexcontacts.Contact
```

That will create this ActionScript class:

```
package  {
   [Bindable]
   [RemoteClass(alias='grails.demo.flexcontacts.Contact')]
   public class Contact {
      public function Contact() {
         // constructor
      }
      public var id:int;
      public var address:String;
      public var city:String;
      public var email:String;
      public var firstName:String;
      public var lastName:String;
      public var phone:String;
      public var state:String;
      public var zip:String;
   }
}
```

11. Edit grails-app/conf/BootStrap.groovy and add some test data.

```
import grails.demo.flexcontacts.Contact
class BootStrap {
   def sessionFactory
   def init = { servletContext ->
      createContacts()
      sessionFactory.currentSession.flush()
   }
   private void createContacts() {
      if (Contact.count()) {
         println 'Contacts already exist'
      }
      else {
         println 'Inserting sample data in table CONTACT...'
         createContact 'Christophe', 'Coenraets', '275 Grove St', 'Newton', 'MA', '02476', '
         createContact 'John', 'Smith', '1 Main st', 'Boston', 'MA', '01744', '617-219-2001'
         createContact 'Lisa', 'Taylor', '501 Townsend st', 'San Francisco', 'CA', '', '415-
         createContact 'Noah', 'Jones', '1200 5th Avenue ', 'New York', 'NY', '', '212-764-2
         createContact 'Bill', 'Johnson', '1345 6th street', 'Chicago', 'IL', '', '', 'bjohn
         createContact 'Chloe', 'Rodriguez', '34 Elm street', 'Dallas', 'TX', '', '415-534-7
         createContact 'Jorge', 'Espinosa', '23 Putnam Avenue', 'Seattle', 'WA', '', '', 'je
         createContact 'Amy', 'King', '11 Summer st', 'Miami', 'FL', '', '', 'aking@mail.com
         createContact 'Boris', 'Jefferson', '222 Spring st', 'Denver', 'CO', '', '415-534-7
         createContact 'Linda', 'Madison', '564 Winter st', 'Washington', 'DC', '', '', 'lma
      }
   }
   private void createContact(String firstName, String lastName, String address, String city
         String state, String zip, String phone, String email) {
      new Contact(firstName: firstName, lastName: lastName, address: address, city: city,
            state: state, zip: zip, phone: phone, email: email).save(failOnError: true)
   }
}
```

12. Start the server.

```
$ grails run-app
```

13. Navigate to http://localhost:8080/FlexContacts/Main.mxml.
The Flash page should load after the webtier mxmlc compiler compiles the swf.

14. Experiment with the application.

- if you leave the search box empty all users will be returned
- you can also search by full or partial name
- double-clicking a name in the result list displays the edit form for that person
- making changes to a person's contact data should update the search results
- creating or deleting a contact doesn't update the search results - re-run the search to refresh
- you can sort the results by clicking the First Name and Last Name column headers

15. Verify that ActionScript isn't browseable.
Attempt to open http://localhost:8080/FlexContacts/Contact.as in a browser - you should get a 403 error page even though it's in the web-app folder and .mxml files load. This is because ActionScript files are explicitly blocked, as are .mxml files in production mode (which we'll see in a bit).

16. Build and deploy a war file.
Shut down the development mode application (using CTRL-C).
Configure precompilation in `grails-app/conf/Config.groovy`:

```
grails.plugin.flex.precompileMxml.enabled = true
grails.plugin.flex.precompileMxml.files = ['Main.mxml']
grails.plugin.flex.precompileMxml.htmlWrapper.create = true
grails.plugin.flex.mxmlc.contextRoot = 'FlexContacts'
```

The last line configuring the `contextRoot` attribute is required since `run-war` deploys with the same context as in development, but in general this should be configured with the actual value of the context root.
Start the app using a war in production mode:

```
$ grails prod run-war
```

Note that the output indicates that Main.mxml gets precompiled and that a GSP wrapper is created for it:

```
[echo] Precompiling MXML Main.mxml
 . . .
[echo] Generating Main.gsp
```

17. Navigate to http://localhost:8080/FlexContacts/Main.gsp.
The Flash page should load very quickly since the swf is already compiled. The functionality should be the same as before when testing with Main.mxml.

18. Verify that ActionScript and MXML aren't browseable in production mode.
Attempt to open http://localhost:8080/FlexContacts/Contact.as in a browser - you should get a 403 error page even though it's in the web-app folder. Also attempt to open http://localhost:8080/FlexContacts/Main.mxml in a browser - you should get a 403 for that too. This is because ActionScript and MXML files are explicitly blocked in production mode - only .swf files can be loaded.