Crudo Plugin.

Documentation.

1 Introduction.

Crudo is an eclipse code generation plugin project. Crudo is aimed to produce the web code and backing artifacts of J2EE Applications. Technologies used include Java Server Faces, Facelets, Hibernate Entity Manager, JPA 3.0. Crudo is composed by a simple web framework, a reflection tool to extract the information from the domain objects and the generator plugin that generates the code.

The tool is in its early development cycle, even thought,it might be quite useful for prototyping.

Master lines.

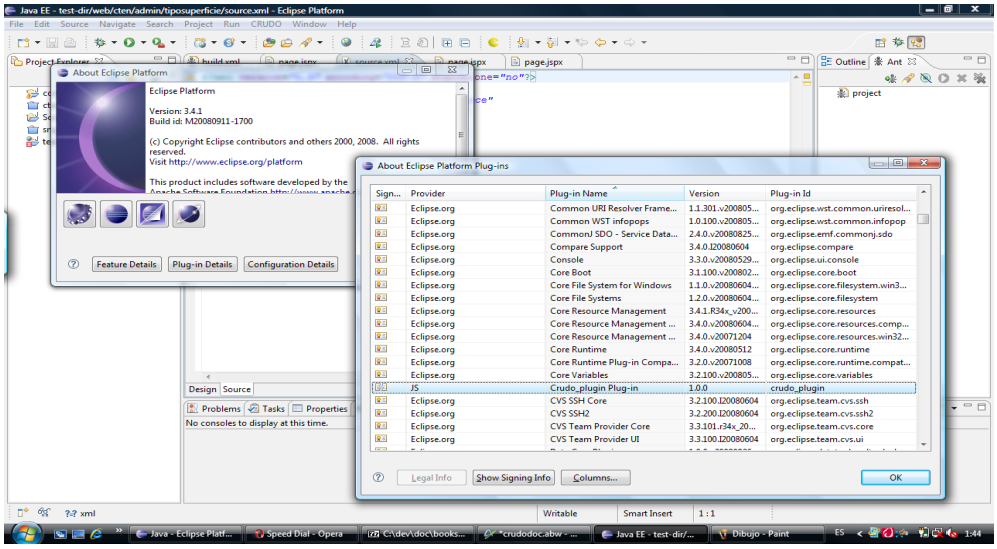
- Model Driven.
- Flexible, and adaptable.
- Code Generation.
- "Reasonable" amount of code.

1 Plugin Tool.

Install.

Copy the crudo-plugin-xxxx.jar into your eclipse plugins directory and restart eclipse. Crudo requires eclipse 3.3 or superior.

Once installed, you can check the installation in help > about eclipse platform > plug-in details.

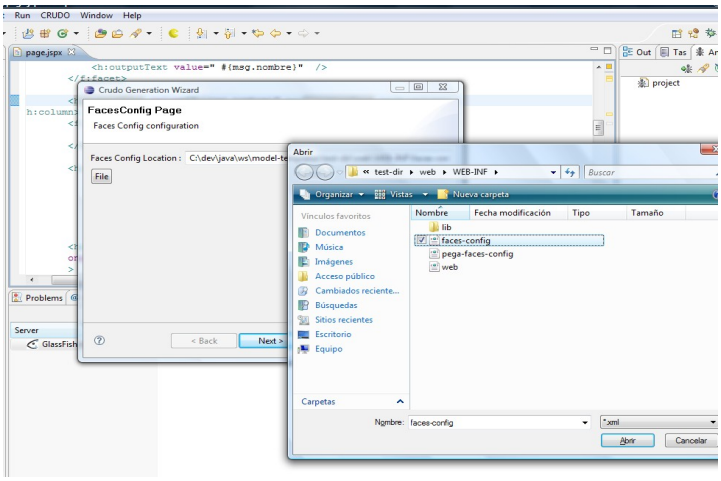


Usage.

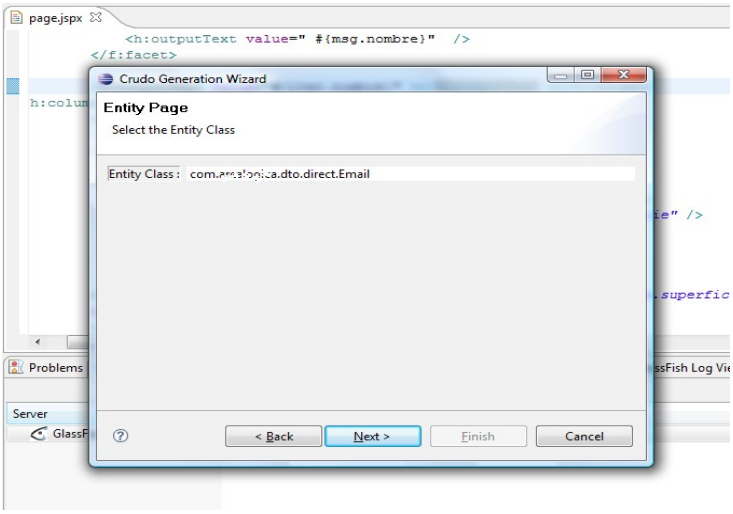
Crudo uses model generated or hard coded entities as input source. Usually you would select the entity, the faces-config file and your output directory.

Generation of an entity Code.

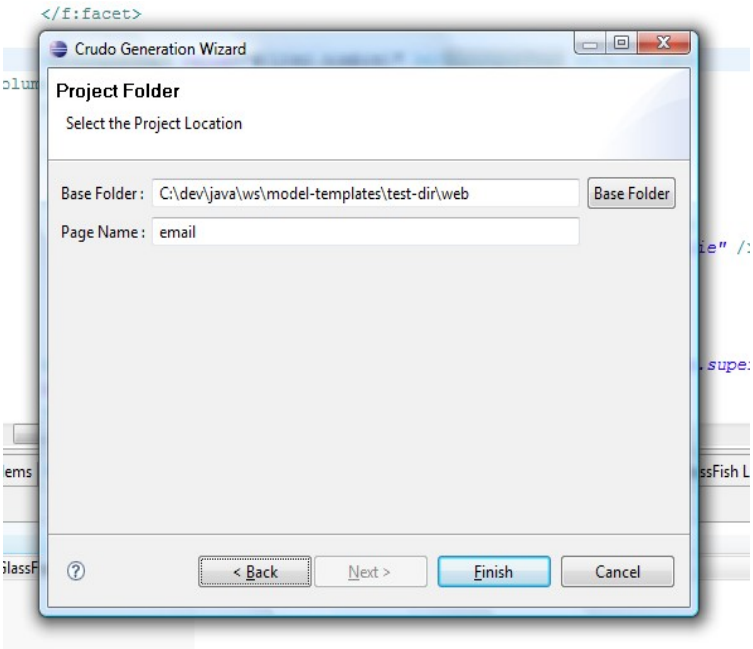
1. Select your faces-config.xml file.



2. Select the entity class name (fully qualified).



3. Select the target folder name.

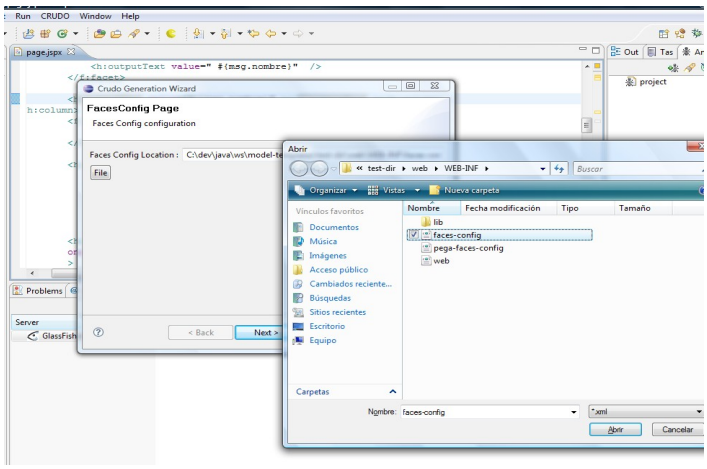


4. Don't forget to refresh your project folder.

Customization of a previously generated Entity.

Crudo uses a file called source.xml, which handles your form data. You can edit that file by hand (order the fields, add or remove fields, add combos, links, etc..). It is possible to describe additional functions.

Once you've edited your source.xml file, launch the crudo generator again with the same entity in the same folder. Your xhtml should reflect your changes.



Crudo copies the default facelets template and css file to your web application folder. Once they are written Crudo don't overwrite that files. Same applies to the source.xml file.

Ordering Fields.

To modify the order of the generated fields, order the fields position in your source.xml and generate again.

Groups.

Use the <group> tag to display logical compositions of fields. It is possible to group fields from the search and edition forms. Logical grouping can be modified with the css properties.

Combos.

Follow the examples to generate combos. Crudo supplies help to feed the combo data.

1 Code Conventions.

Entities.

All domain JPA classes must implement Serializable interface and follow the Java Beans Code conventions(public constructor without arguments, setters, getters, ...). This is an example of a simple JPA Class.

```
@Entity
public class TipoDocumento implements java.io.Serializable  {

		@Id
		@GeneratedValue(strategy=GenerationType.AUTO)
		private Integer id;

		@Column
		private String nombre;

		@Column
		private String clave ;


		.....


		public Integer getId() {
			return id;
		}

		public void setId(Integer id) {
			this.id = id;
		}


		public String getNombre() {
			return nombre;
		}

		public void setNombre(String nombre) {
			this.nombre = nombre;
		}



		public String getClave() {
			return clave;
		}

		public void setClave(String clave) {
			this.clave = clave;
		}

}
```

Nested entities.

Entities must provide a way to initialize its nested objects. The easiest way to do this is to check for null values in your getter methods. You can also override the default function, and handle your nested persistent classes in any other way. Ensure that you have properly set the cascade attribute.

```
public Nacionalidad getNacionalidad() {
        if(nacionalidad == null) nacionalidad = new Nacionalidad();
        return nacionalidad;
}
```

A nested object declaration in your source.xml should like this.

```
<field>
        <name>direccion.codigoPostal</name>
        <type>java.lang.Integer</type>
</field>
```

Once, you have customized the source.xml file run again the wizard.

1 Web application

Usage.

Crudo comes with a bundled web framework. The framework provides the minimal services needed to run the application.

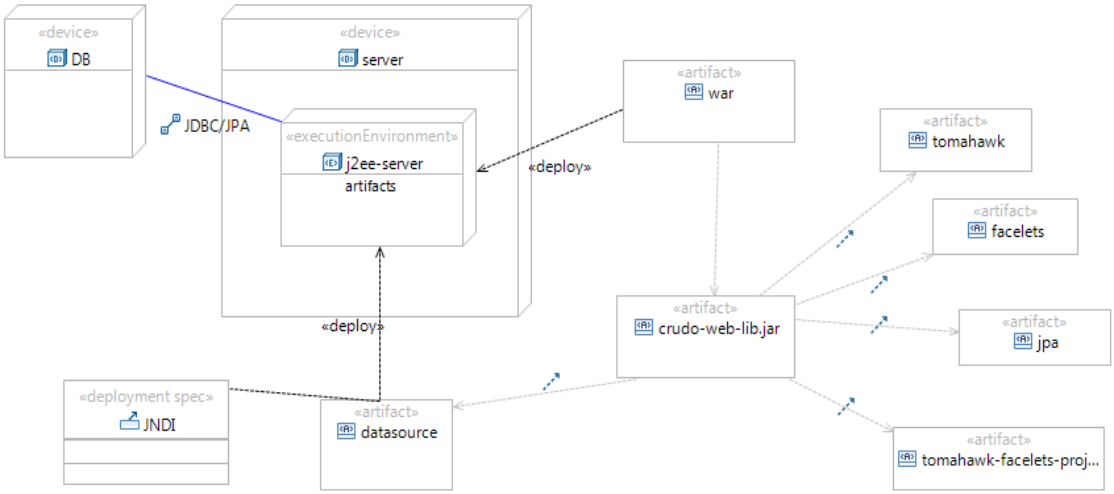
You can use crudo in several ways.
- You can use the generator tool with the provided web framework, overriding the default implementation when needed.
- You can use the tool to generate part of the web code, and use it with your own code.
- You can implement your own framework and use generated web code.

The crudo web framework, is a simple framework focused on extensibility. It should be easy to build different web implementations, or use different persistence services on top of it. You'll have to override different methods to create specific operations.

Web app. Configuration.

Libraries and dependencies.

- Crudo web lib.
- JSF implementation (usually provided by your application server).
- Facelets.
- Hibernate Entity Manager. Other JPA implementation (Top-link, Open JPA should work fine).
- Tomahawk
- Facelets-Tomahawk project.



JPA Configuration.

Persistence unit.

Crudo needs a persistence unit, by default it should be named "default". Future implementations should allow to customize it.

```
        <persistence-unit name="default" transaction-type="JTA">
      <provider>org.hibernate.ejb.HibernatePersistence</provider>
      <jta-data-source>jdbc/DefaultDS</jta-data-source>
       <properties>
                              <property name="hibernate.show_sql" value="true"/>
                              <property name="hibernate.format_sql" value="true"/>
                              <property name="hibernate.use_sql_comments" value="true" />
                              <property name="hibernate.max_fetch_depth" value="3"/>
                              <property name="hibernate.dialect"
value="org.hibernate.dialect.PostgreSQLDialect"/>
                              <property name="hibernate.transaction.manager_lookup_class"
value="org.hibernate.transaction.SunONETransactionManagerLookup" />
                              <property name="hibernate.session_factory_name"
value="HibernateSessionFactory" />
                              <property name="hibernate.generate_statistics" value="true"/>
                              <property name="hibernate.cache.use_second_level_cache"
value="false"/>
                              <property name="hibernate.hbm2ddl.auto" value="update" />
   </properties>
 </persistence-unit>
```

web.xml.

Usually you should reference your persistence unit or your JTA DataSource in the web xml file. Depending on the server runtime, additional deployment descriptors could be needed as usual.

```
<resource-ref>
  <description>DB Connection</description>
  <res-ref-name>jdbc/DefaultDS</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
```

You should also configure the facelets servlet, tomahawk extensions filter and faces context params. Check the documentation from the facelets site.

Faces-config.xml

You must declare the facelet view handler, message bundles, and other application data .

Crudo generates your bean faces-config configuration information. A refresh it's usually needed after the generation process.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<faces-config xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="1.2"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-facesconfig_1_2.xsd">
        <!-- from project setup -->
        <application>
                <view-handler>com.sun.facelets.FaceletViewHandler</view-
handler>
                <locale-config>
                        <default-locale>en</default-locale>
                        <supported-locale>es</supported-locale>
                </locale-config>
                <message-bundle>messages</message-bundle>
                <message-bundle>exception</message-bundle>
        </application>
```

Properties

Default template uses a file called messages.properties. When you generate an entity web code you'll find a file in the generated folder (msg.properties) suggesting values. You can include that file or copy and paste that values in the generic messages.properties.

Web files.

In the distribution archive you will find some images used by the default template. copy them to your web application folder. Css file and default template are created

by the pluggin. You can modify templates, css, and images to fit your desired designs.

Internationalization.

Crudo helps you with your resource bundles. By default, the target application should be fully internationalized using the i18n mechanism. Crudo performs two actions :

1. Declares a link to the default resource bundle (messages.properties).
2. Suggest the field names value in you working folder (msg.properties).

Take the values from the msg.properties in your recently generated folder, check them and paste them in the application messages.properties.

The default facelets template has all its values internationalized.

1 Code Examples.

Source.xml Schema.

Updated (19/01/2009).

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://crudo.sourceforge.net/source"
		xmlns:tns="http://crudo.sourceforge.net/source"
		elementFormDefault="qualified"
		xmlns:Q1="http://crudo.sourceforge.net/source">

  <element name="funcion" type="Q1:funcion"></element>

  <complexType name="funcion">
		<sequence>
			<element name="name" type="string" maxOccurs="1"
				minOccurs="1">
			</element>
			<element name="fields" maxOccurs="1" minOccurs="1"
				type="Q1:fields">
			</element>
		</sequence>
		<attribute name="search" use="optional" type="boolean">
		</attribute>
		<attribute name="list" type="boolean" use="optional"></attribute>
		<attribute name="create" type="boolean" use="optional"></attribute>
		<attribute name="read" type="boolean" use="optional"></attribute>
		<attribute name="udpdate" type="boolean" use="optional"></attribute>
		<attribute name="delete" type="boolean" use="optional"></attribute>
  </complexType>

  <complexType name="fields">
		<sequence>
			<element name="field" type="Q1:field" maxOccurs="unbounded"
			minOccurs="1">		</element>
		</sequence>
  </complexType>


  <complexType name="field">
		<sequence>
			<element name="name" type="string" maxOccurs="1"
				minOccurs="1">
			</element>
			<element name="type" type="string" maxOccurs="1"
				minOccurs="1">
			</element>
		</sequence>
		<attribute name="display" use="optional">
			<simpleType>
				<restriction base="string">
					<enumeration value="combo"></enumeration>
				</restriction>
			</simpleType>
		</attribute>
```

```
            <attribute name="idField" type="string" use="optional"></attribute>
            <attribute name="nameField" type="string" use="optional"></attribute>
            <attribute name="className" type="string" use="optional"></attribute>
  </complexType>
</schema>
```

Source.xml Example.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<funcion search="false" unique="true"
xmlns:xsi="http://crudo.sourceforge.net/source"
xsi:noNamespaceSchemaLocation="./crudo.xsd"
>
        <name>persona</name>
        <fields>
                <field>
                        <name>nombre</name>
                        <type>java.lang.String</type>
                </field>
                <field>
                        <name>apellido1</name>
                        <type>java.lang.String</type>
                </field>
                <field>
                        <name>apellido2</name>
                        <type>java.lang.String</type>
                </field>
                <field>
                        <name>fechaNacimiento</name>
                        <type>java.util.Date</type>
                </field>

                <!-- Direcciones -->


                <field>
                        <name>direccion.calle</name>
                        <type>java.lang.String</type>
                </field>
                <field>
                        <name>direccion.numero</name>
                        <type>java.lang.Integer</type>
                </field>
                <field>
                        <name>direccion.poblacion</name>
                        <type>java.lang.String</type>
                </field>
                <field>
                        <name>direccion.provincia</name>
                        <type>java.lang.String</type>
                </field>
                <field display="combo" idField="id" className="Provincia"
nameField="nombre" >
                        <name>provincia</name>
                        <type>com.some.company.Provincia</type>
                </field>
                <field>
                        <name>direccion.codigoPostal</name>
                        <type>java.lang.Integer</type>
                </field>

                <!-- Email -->

                <field>
                        <name>emailPrincipal.email</name>
```

```
                        <type>java.lang.String</type>
                </field>


                <!-- Telefonos -->

                <field>
                        <name>telefonoPrincipal.numero</name>
                        <type>java.lang.String</type>
                </field>



                <field>
                        <name>telefonoMovil.numero</name>
                        <type>java.lang.String</type>
                </field>


                <!-- Pagina -->

                <field>
                        <name>paginaPrincipal.pagina</name>
                        <type>java.lang.String</type>
                </field>

        </fields>
</funcion>
```