

THE OpenToro TEAM.

OpenToro Tutorial.



OpenToro. Reference.

- 1.- Introduction.
 - 2.- OpenToro 'Programming': XML metadata.
 - 3.- Architecture and TagLib.
 - 4.- References.
-

1.- Introduction.

OpenToro is a Web Database Publisher, a tool that allows us developing database-driven web applications in an agile and automatic way. Using OpenToro simply means to forget coding countless SQLs and JSPs every time we want to implement a web application with database access.

With Opentoro you will be able to easily:

- ☞☞ Listing Database Tables.
- ☞☞ Visualizing Records.
- ☞☞ Generating Forms for inserting, modifying and deleting records.

OpenToro Works with any SQL-92 compatible database, but if you want to use advanced features of some databases you can use (or develop) specific SQL Engines. OpenToro implements SQL Engines for the following databases:

- ☞☞ MySql.
- ☞☞ Oracle.
- ☞☞ Access.
- ☞☞ SQL-Server.
- ☞☞ HSqlDB.

OpenToro's web is: <http://opentoro.sourceforge.net>

This version have made a strong effort implementing advanced Ajax technology, like

- ☞☞ Combo's reloading (with N-Dependences).
- ☞☞ TextSuggest fields.
- ☞☞ Ajax record's search.
- ☞☞ Embedded loading of forms, records and listings.

We have incorporated a configured HsqlDB database inside the OpenToro distribution, so you can test OpenToro without any additional configuration.

2.- OpenToro 'Programming': XML metadata.

OpenToro works in such a way that is able to manage any database's table knowing its properties. The information that describes the structure of a table is



called metainformation, and OpenToro saves it in XML files in the directory 'WEB-INF\metadatos'.

Here is an example of metadata file for a table called 'SCOPES' that have two fields: 'CodScope' and 'Name'.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<MetaData>
  <TableDescription>
    <TableGeneral>
      <TableName>SCOPES</TableName>
      <TableDesc>SCOPES</TableDesc>
      <TableIndexField>CODSCOPE</TableIndexField>
    </TableGeneral>
    <FieldsDesc>
      <FieldDesc>
        <FieldName>CODSCOPE</FieldName>
        <FieldAlias/>
        <FieldDescription>CODSCOPE</FieldDescription>
        <FieldType>num</FieldType>
        <!--FieldReference codMetaview=""--></FieldReference-->
        <FieldRequired>S</FieldRequired>
        <FieldLength>6</FieldLength>
      </FieldDesc>
      <FieldDesc>
        <FieldName>NAME</FieldName>
        <FieldAlias/>
        <FieldDescription>NAME</FieldDescription>
        <FieldType>text</FieldType>
        <!--FieldReference codMetaview=""--></FieldReference-->
        <FieldRequired>S</FieldRequired>
        <FieldLength>100</FieldLength>
      </FieldDesc>
    </FieldsDesc>
  </TableDescription>
  <MetaViews>
    <MetaView cod="100">
      <MetaViewGeneral>
        <MvDescription>SCOPES</MvDescription>
        <MvTitle>SCOPES</MvTitle>
        <MvType/>
        <MvOrderField orderType="ASC">CODSCOPE</MvOrderField>
        <MvComboField>CODSCOPE</MvComboField>
        <MvGroupByExpr/>
      </MetaViewGeneral>
      <MetaViewList>
        <Field>CODSCOPE</Field>
        <Field>NAME</Field>
      </MetaViewList>
      <MetaViewRecord>
        <Field>CODSCOPE</Field>
        <Field>NAME</Field>
      </MetaViewRecord>
      <MetaViewForm>
        <Field>CODSCOPE</Field>
        <Field>NAME</Field>
      </MetaViewForm>
    </MetaView>
  </MetaViews>
</MetaData>
```

As we can see the XML begins with a MetaData node that includes the rest of nodes. Next we have two parts, a defined one in the node TableDescription and another defined one in the node MetaViews, the first part is good for describing the



table in a general way, that is to say, name of the table, description, fields, etc... and the second part is where there are defined the different metaviews for the table in question.

With respect to the first part we have following nodes:

TableName: In this node we indicate the name of the table.

TableDesc: In this node we indicate a description for the table.

TableIndexField: In this node we indicate the field that forms the primary key of the table.

Later we can find the nodes with the description of the table's fields (sometimes referred as metafields). Let's take a look:

FieldName: Field's name.

FieldAlias: Alias name to be used in the SQL for that field. It is useful when the metacampo has a great quantity of characters or it is affected by a function that uses simple quotation marks. For example, if we want to show the name and the last name of a person like a single field we could use the oracle function,

`concat(concat(Surname,' '), Name)`, and assign the alias CompleteName.

FieldDescription: Field's name that it's going to be viewed in listings and forms.

FieldType: It is the type of the field. It is related intimately with the type of the field of the physical object (table's field) in the database. The possible values are:

- **text:** text type (for example VARCHAR in MySql)
- **num:** numeric type (f.e. INT in MySql)
- **codn:** numeric type that it is too a foreign key in another table. The metaview's code referenced must be specified in the node FieldReference.
- **codt:** text type that it is a foreign key in another table. The metaview's code referenced must be specified in the node FieldReference.
- **idRef:** field of text type that is foreign key of another table, although not pointing to the primary key of the table master, but to a field, also of type text, with the restriction of being unique. Therefore, a field metaprogrammed with the type idRef already have the value of what is wanted to show, without necessity of making a query to obtain the indexed value. The metaview's code referenced must be specified in the node FieldReference. We
- **memo:** text type of big size (f.e. LONGTEXT in MySql). To put a field with type memo means that will be shown like a TextArea in the HTML forms.
- **date:** date type. We recommend working with codn and codt instead of idRef, because the textual referentes causes problems when you change the master table.
- **bool:** boolean type (f.e., VARCHAR(1) with values 'Y' or 'N' in MySql).
- **url:** text type for URL. The data saved in this field is later generated like a HTML link.
- **file:** text file that will store the path to a file. The file is uploaded by OpenToro (using a form).



- **img:** text file that will store the path to a image file. The file is uploaded by OpenToro (using a form). OpenToro later generate in the HTML a IMG tag pointing to the image file.
- **combodependent:** is very similar to codn type, but the value of this field depends on a field that must be metaprogrammed just above. In the Forms, this field will appear like a combo, that will be automatically recharged each time that his combo father change.
- **Suggest:** is very similar to codn too, but this field will appear like a input field that will be autocompleted (retrievings completions) while you are writing.

FieldReference: If the field metaprogrammed is a foreign key, we must to specify the table referenced using the 'codMetaView' attribute. We must to specify too the field that will be used in the visual substitution (usually named 'fusion') in the listings and record contents visualization. If we are generating a form, OpenToro will show a combo ('select' tag in HTML) with the sets of values of the referenced table (in this case the field used in the fusion must to be indicated in the node 'MvComboField'.

FieldRequired: This node indicates if the field is required. This is special useful in forms generation, since OpenToro generates validation code for this fields.

FieldLength: Field's length allow in the database. OpenToro's forms validate this too.

With respect to the XML's second part, we must begin describing each metaview with the node <MetaView cod = "1001"> in whose attribute (cod) indicates the metaview's code (that function like an ID). We have the following nodes:

Grouping in the **MetaViewGeneral** node:

MvDescription: In this node we can specify a description for the metaview. This description should be as precise as possible, since we can have several metaviews of the same database table, and this description will be good to distinguish them.

MvTitle: In this node, we indicate the metaview's representative text, which will appear in the listings, and forms.

MvOrderField: In this field we indicate the order in which will appear the records' listings of the metaview, we also have the attribute orderType where we can indicate if the order should be "ASC" or "DESC".

MvComboField: It indicates the field that will be shown if you want to generate a HTML combo of this metaview. In what cases will it be contemplated using this field?

- ?? When we want to take out a HTML select (combo) from a page web and let us make a call to the OpenToro to the method getCombo(int)
- ?? When we want to take out an empty or filled form of a metavista (let's name it 'metaview A'), calling to the method getForm (...). If this metaview has some field that is a foreign key (referencing to another metaview (let's name



it 'metaview B')), the field indicated in the metaprogramming (MvComboField) of the metaview B, will be the one shown in the select that will be obtained automatically in a form of the metaview B.

MvGroupByExpr: This node allows including a 'group by' clause to the SQLs generated for this metaview.

Then we can observe that we have three very similar nodes: MetaViewList, MetaViewRecord and MetaViewForm. In these nodes we indicate the fields that we want to show respectively in a listing, a record's visualization or a form, the order in which they will appear corresponds with the order that will have in the HTML.

For example, for listings we can show simply the next fields:

```
<MetaViewList>
  <Field>TITLE</Field>
  <Field>AUTHOR</Field>
  <Field>CODSCOPE</Field>
</MetaViewList>
```

For record's content we can erase the CODNEW field, since it is an autoincremental field, and is not useful for the user.

NOTE: For FORMS usually we must need to specify hidden fields (typical HTML markup like this: <input type="hidden"....). For this situation, we can use the hidden attribute:

```
<Field hidden="Y">CODSCOPE</Field>
```

Once we have finished modifying the XML files, we must return to the OpenToro's main page, and next to re-establish the database connection. With this action, OpenToro will recharge the metadata contained in the XML files.

XML validation.

Frequently we need to modify the XML files generated by OpenToro. If we do so, is very important validating the modified XML files.

Doing it is very easy. We can use the 'DEHESA' utilities in the OpenToro web application, clicking in the 'XML validation' option. If we don't have a database connection, OpenToro shows you the form for connecting.

Under the form we can see a XML files listing. These files are the files contained in the metadata directory (initially '/metadatos' directory).



The screenshot shows the OpenToro web interface. On the left is a navigation menu with items: OEHEDA, Validate XML, Metadata from Database Objects, and OpenToro. The main content area is titled 'OEHEDA' and contains a section for 'Connection's parameters'. This section includes a form with fields for Driver (jdbc:mysql:Driver), User of BBBB (toro), Password (toro), Url (jdbc:mysql://10.229.213.34:3306), Source (MySQL), and Metadata directory (metadatos). Below the form, there is a note about the file 'ConnectionFactory.properties' and a button 'Establish new connection'. A red message states 'The connection has established.' and explains that the established connection doesn't load the XSLT transformation and metadata. Below this is a section titled 'VALIDATION OF XML METADATA FILES' with a 'Select a file to validate' dropdown menu. The dropdown shows 'news_1002.xml' and 'scopes_1001.xml'.

If we click the XML file's name, OpenToro will validate the file:

The screenshot shows the 'Validation of XML file' section. It displays the result of validating the file 'NEWS_1002.xml' with XSchema as 'CORRECT'. Below this, it shows the result of the rest of the validations of the file 'NEWS_1002.xml' as 'CORRECT'. A message states 'You can continue validating available XML files:'. Below this is a 'FileName' section with a dropdown menu showing 'news_1002.xml' and 'scopes_1001.xml'.

OpenToro do two validations: an XSchema validation, and a set of validations using the database as metadata source.

If the XML file is not valid, OpenToro will show detailed messages about the errors:

The screenshot shows the 'Validation of XML file' section. It displays the result of validating the file 'NEWS_1002.xml' with XSchema as 'CORRECT'. Below this, it shows the result of the rest of the validations of the file 'NEWS_1002.xml' as 'INCORRECT'.

THE OpenToro TEAM.

OpenToro Tutorial.



MetaData\MetaViews\MetaView codMetaView=1002\MetaViewList\Field\CODPROYECTO

You can continue validating available XML files:

FileName

[news_1002.xml](#)

[scopes_1001.xml](#)



Reference Tables.

Usually there are database tables containing data that almost never change (for example the set of States of USA).

OpenToro have an inner Reference Tables Cache, for improving performance. In our example, the 'Scopes' table could be a reference table.

We can specify the name of the reference tables in the 'TORO_TABLASREF.ref' file, which it will be in the metadata directory.

This is not an obligatory file, we don't need its existence.

In our example, the TORO_TABLASREF.ref file contains this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<MetaData>
  <TablesReference>
    <TableName>SCOPES</TableName>
  </TablesReference>
</MetaData>
```



3.- Architecture and TagLib.

The OpenToro architecture is composed by 3 sub-architectures:

1. A Model 2 architecture for managing the user requests and generating responses. This is the Web Application.
2. A Pipeline architecture for generating contents (XML based). These contents are transformed by XSLT, generating finally HTML.
3. A Reflective architecture for managing the content's tables.

The OpenToro architecture has been designed for making easy developing web applications that put information on the web, taking as source a database.

OpenToro makes trivial generating listing of database's tables, visualizing records contents and making forms for database's tables

Adding a new JSP in the OpenToroWeb projects consist in putting it in the path that you want and linking it with a URL like this:

[*FrontController?action=Static&url=/example_folder/example.jsp*](#)



TagLib: Listings.

```
<opentoro:List      metaview="[num]"      order="[asc|desc]"      page="[num]"
pagination="[true|false]" range="[num]" template="[num]" type="[List|Combo|Filters]">
  (<opentoro:Filter column="[txt]" operator="[txt]" value="[txt]"/>) *
  (<opentoro:FilterByRequest enabled="[true|false]"/>) ?
  (<opentoro:HavingTextualFilter filter="[txt]" />) *
  (<opentoro:TextualFilter filter="[txt]" />) *
  (<opentoro:XsltParameter parameter="[txt]" value="[txt]" />) *
</opentoro:List>
```

opentoro:List

Use this tag for show lists.

These are the tag's attributes:

- metaview: number of the metaview that we want to list.
- order: order's type (asc or desc).
- page: page number that we want to show.
- pagination: Do you want to make pagination in the list? Values: true or false.
- range: number os records per page that we want to list.
- template: xslt's code that we are going to use to generate the HTML. OpenToro have a set of XSLT ready for this.
- type: Witch type of listing do you want to show? This attribute must to have one of this values: list (for normal listings), combo (all the values are elements of a combo), filters (generating all the filters associated with the metaview (foreign keys)).

```
<opentoro:List metaview="1001"
template="<%=HelperTemplate.NORMAL_LIST%>" page="-1" range="10"
pagination="false" type="List">
  . . .
  . . .
  . . .
</opentoro:List>
```

opentoro:Filter column="[txt]" operator="[txt]" value="[txt]"

With this method you add a condition for the SQL query that OpenToro will execute before generating the listing (the SQL query takes the data that will be listed).

- Column: Column name of the condition
- operator: Filter's type ('=', '<>', '>', etc...)
- value: Filter's value.

THE OpenToro TEAM.

OpenToro Tutorial.



```
<opentoro:Filter column="country" operator="=" value="spain" />
```

(The listing will only include the records that have the value 'spain' in the column 'country').

opentoro:FilterByRequest enabled="[true|false]"

You can add the request's parameters-values like filters for the listing. OpenToro automatically check if the parameters' names are fields of the metaview, putting only the corrects ones.

```
<opentoro:FilterByRequest enabled="true" />
```

opentoro:TextualFilter filter="[txt]"

This tag is used to add a condition that will be placed in a textual way in the where sentence of the SQL query. This SQL query takes the data that will be listed.

```
<opentoro:TextualFilter filter="age = 21" />
```

opentoro:HavingTextualFilter

This tag is used to add a having condition that will be placed in a textual way in the where sentence of the SQL query.

opentoro:XsltParameter

This tag is used for adding a xslt parameter to the listing.

- parameter: name of the parameter in the xslt.
- value: value of the parameter.

```
<opentoro:XsltParameter parameter="isAjax" value="S" />
```



TagLib: Record's Content and Forms.

Record: Shows a record of a table.

```
<opentoro:Record element="[num]" metaview="[num]" template="[num]">
  (<opentoro:XsltParameter parameter="[txt]" value="[txt]" />)*
</opentoro:Record>
```

It's very similar to the tags that we have just described. The basic element is the tag Record, and the next attributes:

- element: value of the primary key of the table, basically a number.
- Metaview: value of the metaview (that points to a table).
- Template: number of the template that we are going to use for generating de record.

Form: This tagis used for showing Empty or Filled forms from a metaview. This tag generate a HTML form.

```
<opentoro:Form action="[txt]" element="" enctype="[txt]" metaview="[num]"
method="[post|get]" name="[txt]" template="[num]" type="[EmptyForm|FilledForm]">
  (<opentoro:HiddenField parameter="[txt]" value="[txt]" />)*
  (<opentoro:XsltParameter parameter="[txt]" value="[txt]" />)*
</opentoro:Form>
```

Attributes:

- action: Action attribute of the <form> html tag.
- enctype: Enctype attribute of the <form> html tag.
- method: Method attribute of the <form> html tag.
- name: Name attribute of the <form> html tag.
- element: If we want to modify a record we must to indicate the cod of that record.
- metaview: Metaview's code from we want to show the form.
- template: Xslt's number that we want to use.
- type: Form's type that we need (EmptyForm or FilledForm).

opentoro:HiddenField

This tag is used for generating hidden fields in the form.

- parameter: Hidden field's name.
- value: value of the hidden field.



FormFields: This tag generates an empty or filled form but without the 'header' html tag of the form, that must to be build manually in the JSP.

```
<opentoro:FormFields          metaview="[num]"          template="[num]"
type="[EmptyForm|FilledForm]">
  (<opentoro:XsltParameter parameter="[txt]" value="[txt]" />)*
</opentoro:FormFields>
```

FormHeader: This tag builds a HTML form, letting to include composed forms.

```
<opentoro:FormHeader    action="[txt]"    element="[num]"    enctype="[txt]"
method="[post|get]" name="[txt]">
  (<opentoro:HiddenField parameter="[txt]" value="[txt]" />)*
  (<opentoro:FormFields />)*
  (<opentoro:Hierarchical />)*
  (<opentoro:List />)*
  (<opentoro:Record />)*
</opentoro:FormHeader>
```

4.- References.

- ~~🔗~~ OpenToro: <http://opentoro.sourceforge.net/>
- ~~🔗~~ MySql: <http://www.mysql.com/>
- ~~🔗~~ Apache Jakarta Tomcat: <http://jakarta.apache.org/tomcat/>
- ~~🔗~~ 'Internet Application Design Using J2EE Design Patterns', URL: <http://www.moisesdaniel.com/wri/desaplj2ee.html>
- ~~🔗~~ 'Content Managemen on Web Sites:' URL: <http://www.moisesdaniel.com/wri/cmws.html>
- ~~🔗~~ 'Reflective Architectures and Code Generation'. URL: <http://www.moisesdaniel.com/wri/racg.html>
- ~~🔗~~ Tecnología HTML: <http://www.w3.org/MarkUp/>
- ~~🔗~~ Tecnología CSS: <http://www.w3.org/Style/CSS/>
- ~~🔗~~ Tecnología XSLT: <http://www.w3.org/TR/xslt>