

oXerp Documentation

1.	Installation	2
2.	Add extra info to an object	3
3.	Close a period .	4
4.	Create a company	5
5.	Create a partner.	6
6.	Customize invoice	7
7.	Customize payment conditions	8
8.	Import-export.	9
9.	Marshalling java/xml .	10
10.	Understand config files	12
11.	Changelog b2 to b3	13

1. Installation

Notes

Directories are noted in ant variable mode ; for instance `${oxerp.home}` stands for a directory you choose to be the main directory for oxerp, for instance "c:/oxerp" or "/oxerp" according to your platform. Be careful to choose 8 characters max under Win98 for directories names. You can build server and client using the "build section" below. You can use directly the prebuild bundles with the "server" and "client" sections

Server

- Install jvm1.3+ : java.sun.com
- Install JBoss3.x : www.jboss.org, download and unzip in `${jboss.home}` (we recommend JBoss but you can use another AppServer)
- Install oxerp-server.jar in `${jboss.home}/server/default/deploy`
- Run JBoss : `${jboss.home}/bin/run`
- You are done with the server-side

Client

- Unzip oxerp-client.zip in `${oxerp.home}`
- Unzip oxerp-client-lib-jdk13 or jdk14.zip in `${oxerp.home}`

Alternatively :

- Download fop.jar (0.20.4+), batik.jar, avalon-framework.jar : www.apache.org, to be able to print, drop it in `${oxerp.home}/lib`
- only if jdk13 : Download saxon.jar : www.apache.org, drop it in `${oxerp.home}/lib`

Modify the run script :

- set the JBOSS_HOME variable to `${jboss.home}` directory
- set the JAVA_HOME variable to jvm home

Login : admin, password : admin

build

- Install ant-1.5 : download from jakarta.apache.org and unzip
- Install XDoclet : www.xdoclet.org
- Use the oxerp-src.zip and the script/build.xml file with the clean-build target

2. *Add extra info to an object*

This is possible for Partner, Product, Invoice. You use the xml-info field. This field can contain any kind of data ; xml is the wished format since you can provide that way structured or semi-structured datas. For instance, in european countries, we can add there the intra-EE VAT number since it is not provided in the general Partner prototype. This info can be used on the client side : for instance, you can get it back to print it in an invoice when you [customize invoice.xsl](#)

3. *Close a period*

The closing of a period is a quite important operation : after that it is impossible to modify anything in the given period or to add new invoices. Choose the working period In the accounting module, int the 'Tools' menu, validate your 'Brouillard' and then close the period in the same menu.

4. *Create a company*

Once logged in, you reach the company creation through main menu/company option. Create your new company just choosing the name and password if you wish. Log in your new company. Then, the easiest way is to import a template company from config directory (See [howto-import-export.xml](#)) For the time being, two templates exist : French plain company, using 2050 forms and French free lance, using 2035 form :

- . independant_2035
- . societe_reel_2050

This procedure makes you up with a consistent accounting map and all default options ready on your company ; visit in the main menu 'Company parameters' to see them. You can do it by hand but quite long and inconsistency source. Finally, we recommend the creation of a sub-directory in config with your company name. In this subdirectory override any file present in default subdir with your own customized files (eg. : invoice.xsl, oxerp_client-config.properties for a societe_reel_2050)

5. *Create a partner*

A partner is a client or a provider or both of them. The only difficulty when creating a partner is to choose the imputation accounts. In a first approach, we would recommend to create an account for this partner (two if he is both a client and a provider). Then you use the same account for the four items required.

6. *Customize invoice*

XSL is the language used to customize documents. Just create a sub-directory with the name of your company below config in your installation tree. Add a customized invoice.xml file in it. You can find default invoice.xml in config/default sub-directory. Here is an example of invoice.xml to be transformed by invoice.xml ; this xml is produced by class InvoicePrintableHelper :

```
<document type='Facture' no='2003-202' date='31 mai 2003' chemin='Z:\Projets\Aj3\src\config\Pascal' echeance='30J
date de facture' mode='Virement' bdc='BC000496'><xml-info>null;null;null;null;</xml-info><destinataire name='PARTNER'
code='P0' nom='Toto' title='Mr' prenom='Toto' libelle='Somewhere' rue='0, street St John' zip='75000'
ville='PARIS'><xml-info>null</xml-info></destinataire><ligne libelle='SomeProduct' product='INTJ' quantite='5.0'
amount='5000' vatRate='0.196'><xml-info>null</xml-info></ligne></document>
```

7. *Customize payment conditions*

- Create a class : to give the forecast payment date (heritance from PaymentConditions) :
`org.oxerp.plugins.pcm.PaymentConditionsNEW_CONDITIONS`
- Add a line in `oxerp_other_*.xml` : `payment.conditions.NEW.CONDITIONS`
- Add some data in `oxerp_societe.xml` in `config/YOUR_COMPANY` directory : `<payment-condition key='payment.conditions.NEW.CONDITIONS' class='org.oxerp.plugins.pcm.PaymentConditionsNEW_CONDITIONS'/>`

8. *Import-export*

In the main menu, you can use Importation/Exportation items. Before that, for export, be sure to choose a correct 'Working period' : only invoices and vouchers in this period will be exported. Once the directory chosen, xml files are imported/exported to it. Try an export to know the list of xml files. A dialog box inform you about failure/success of the operation. Import is not transactional. But if some failure occurs, you can correct the cause and try it again. Datas yet imported will be ignored ; only those which failed on the first time, will be imported.

9. *Marshalling java/xml*

Principles

Oxerp uses a custom marshaller/unmarshaller. The classes of this marshaller are under packages `org.oxerp.xml.*`. For the time being, the marshaller is used only on the client side. This will change in future releases, mainly for import-export use.

Mapping file

The marshaller needs a mapping file to work. This format of the mapping file is compatible with Castor project (`org.exolab.xml`). As a consequence you can use the XDoclet castor tags to generate this file. A section of the mapping file looks like :

```
<class name='org.oxerp.xml.test.ClassTest'>
  <map-to xml='product' />

  <field name='categ' type='java.lang.String' direct='true'>
    <bind-xml name='category' node='attribute' />
  </field>

  <field name='lib' type='string' direct='true'>
    <bind-xml name='libelle' />
  </field>

  <field name='d' type='date'>
    <bind-xml name='date' node='attribute' />
  </field>

  <field name='examples' type='string' collection='collection'>
    <bind-xml name='col-ex' node='element' />
  </field>

  <field name='inner' type='org.oxerp.xml.test.ClassTestInner'>
    <bind-xml name='inner' />
  </field>

  <field name='inners' type='org.oxerp.xml.test.ClassTestInner' collection='collection'>
    <bind-xml name='inners' node='element' />
  </field>

  <field name='paymentConditions' type='org.oxerp.xml.test.ClassTestAbstract'>
    <bind-xml name='payment-conditions' node='element' />
  </field>

  <field name='examplesSet' type='string' collection='set'>
    <bind-xml name='col-ex-set' node='element' />
  </field>
</class>
```

- Tag class, attribute name : the class to marshal
- Tag field, attribute name : a java attribute in the class if direct is set to true, a set/get couple method if not
- Tag field, attribute type : string, java.lang.String, date, java.util.Date, integer, int, double, long, short, boolean, double, float are all default basic types. You can put another class described by the mapping file in the type attribute.
- Tag field, attribute collection : if set to collection a java.util.List is expected, if set to "set", a java.util.Set is expected
- Tag bind-xml : the xml counterpart of the field, xml attribute or xml sub-element

Custom simple types

You can plugin your own simple type. To do that, you add a sub class of `org.oxerp.xml.SimpleType`, defining the method `format`, `parse` and `getClassType`. Then, you add a section "simple-type" section in mapping file (not compatible with Castor mapping file) ; the name attribute defines the virtual simple type used in the "field" tag. See `org.oxerp.xml.test.SimpleTypeDecimal` with `oxerp_mapping.xml` in `oxerp-marshaller.jar` (in the client zip for instance)

```
<simple-type name="decimal" class="org.oxerp.xml.test.SimpleTypeDecimal"/>
```

API calls

```
Mapping m = new Mapping();
try {
    m.loadMapping("oxerp_mapping.xml");

    unmarshaller = new Unmarshaller();
    unmarshaller.setMapping(m);

    Reader reader = new BufferedReader(new FileReader(file));
    Object o = unmarshaller.unmarshal(reader, ClassTest.class);
    ClassTest ct = (ClassTest)o;

    Marshaller marshaller = new Marshaller();
    marshaller.setMapping(m);
    Writer w = new BufferedWriter(new FileWriter(file));
    marshaller.marshal(o, w);
    w.close();
}
```

10. *Understand config files*

Quite a messy point for the time being

- oxerp_(*)(fr-en)(FR-EN).properties : resource bundles for internationalization .
oxerp_other_*.properties : custom translation for company : eg. payment modes and conditions
- - oxerp_client_config.properties : client configuration for a company (Societe) . parametrization of reports (<etats>) in accounting module . <modules> available . misc property : eg. default.vat.rate
From the api, you get back the value through sessionAj.getSocieteConfValue
- oxerp_server_config.properties : connection to oXerp server and main configuration for client, cross-comapnies (frame size, locale...) From the api, you get back the value through SessionAj.getServerResourceString

11. *Changelog b2 to b3*

END-USER FEATURES

- Formatting doc
- xml format import/export : without change b2-b3 : you should be able to reimport as-is you b2 exports

DEVELOPER FEATURES

- castor replaced by a custom marshaller/unmarshaller (smaller and more suitable for oXerp)
- double replaced by class Decimal as a wrapper (numbers are saved as long to avoid rounding problems)

BUGS CLIENT-SIDE CORRECTION

- import invoices before, related avoires after in order not to have them rejected
- NullPointerException in TableRenderer for null datas
- Quantities and unit prices with a negative sign

BUGS SERVER-SIDE CORRECTION

-