# *OFBiz An Insider View*

Prepared By:

Basil Argasosy
Senior Computer Engineering Student
King Fahd University of Petroleum & Minerals (K.F.U.P.M)
September 01, 2005

Contact Information

s208603@kfupm.edu.sa
st208603@ccse.kfupm.edu.sa

or through my personal webpage

# OFBiz : An Insider View

## Introduction:

The OFBiz framework utilizes the common Three-Tier “Layers” Architecture model in all its applications. It has the Data Layer, the Business “logic” layer, and the Presentation “user interface” layer. The Data Layer and the Service layer have their own engines that are responsible for interaction with the layer.

1) Data Model Layer: It represents the database. There is an Entity Engine that is responsible of this layer that includes database connection, data retrieval, data storage…etc. It used the java Generic Delegator class to connect with the database, and it uses the java Generic Value to represent an entity row to be inserted in the database.

2) Business Logic Layer: It represents the logic, or the services provided to the user and performed on the data layer "database”. There can be services of many types like java, SOAP, simple, workflow, etc. and each type of service has its own handler. There is a Service Engine that is responsible for dealing with services, calling the service, etc.

3) Presentation Layer: OFBiz has moved to use "Screens" to represent the OFBiz pages. So, each page should normally be represented as a screen. An OFBiz page consists of many components like headers, footer, appheader,..etc, so when rendering the page, these are all combined in the order they were placed, or included, in the screen.

4) Servlet Container : This is the main servlet that controls all the application “controller.xml” .The controller defines the event handlers and the view handler, type of the services, the location of the views..etc. Web.xml is an important file to configure the main servlet(s) and also to control to tomcat server.

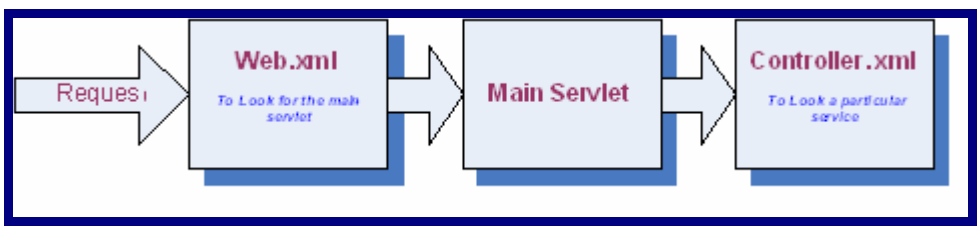


**Figure 1**

## Practical Overview:

Before starting to build our new application, let's have a look inside the OFBiz. Here is the OFBiz application folder on the C drive.

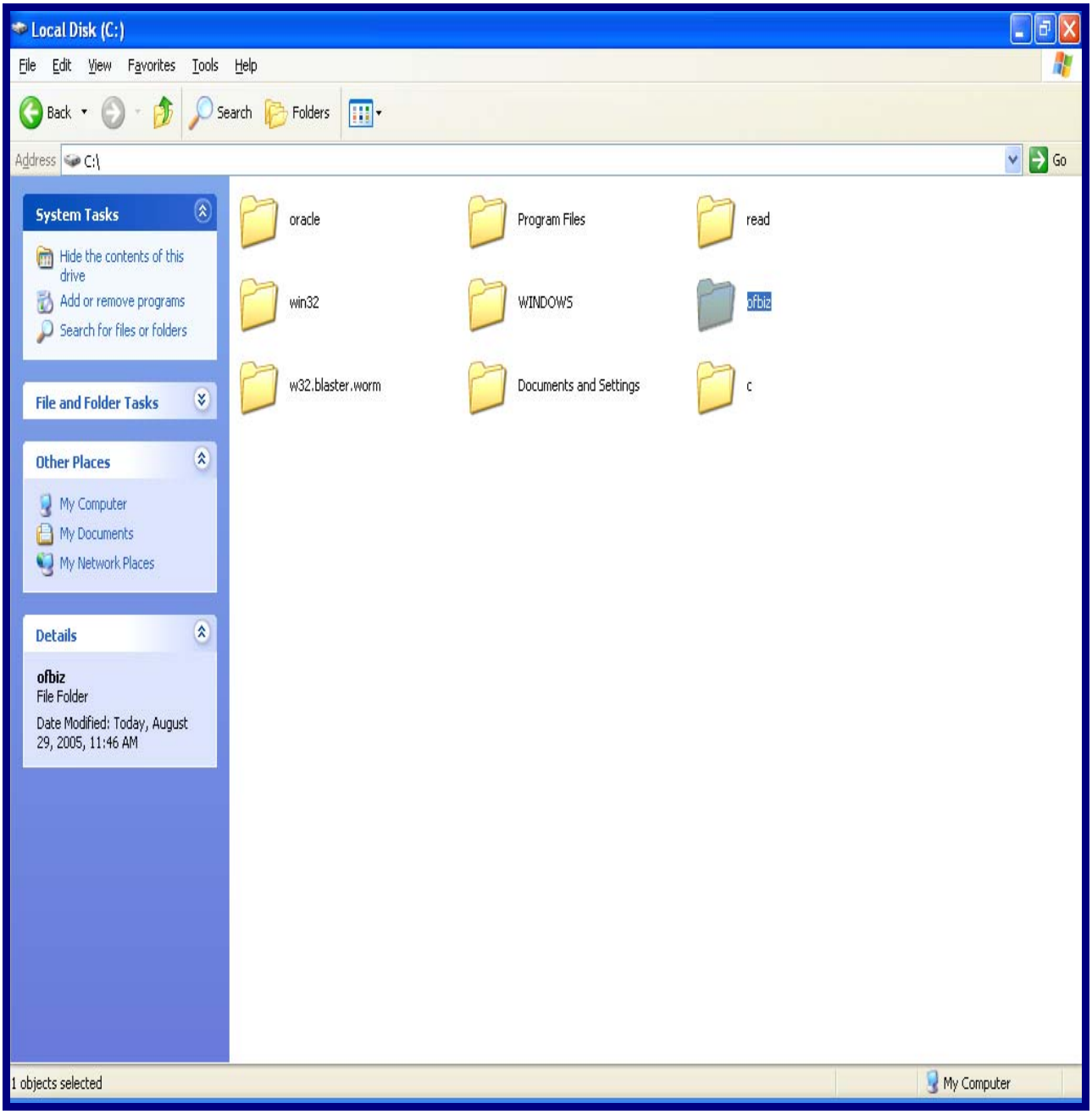


**Figure 2**

Having a look inside the OFBiz folder, we would see the following ：

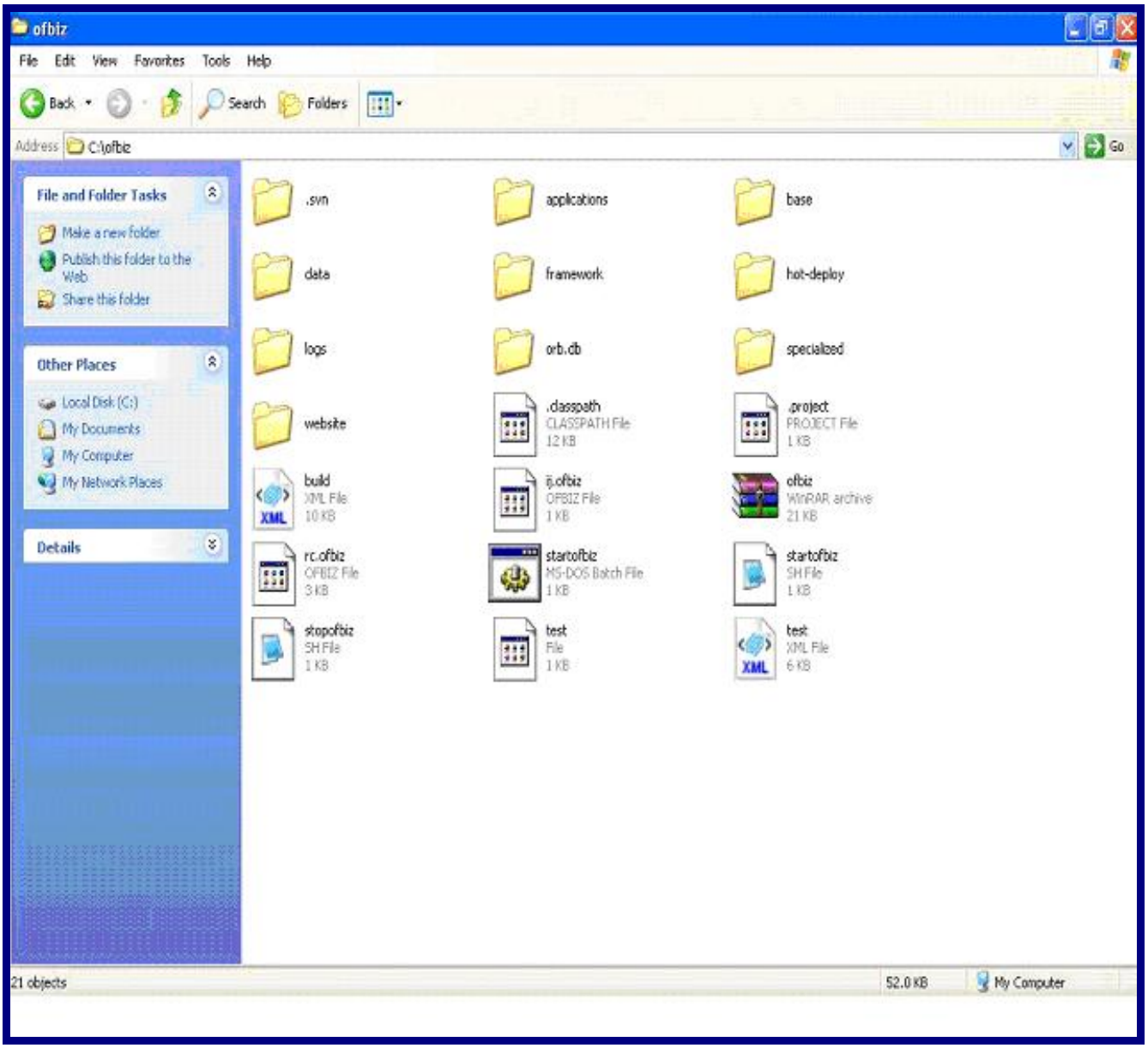


**Figure 3**

These folders are as follows: *[1]*

.svn folder : contains the weekly update batches for the OFBiz framework and applications.

applications folder : contains the application components created with OFBiz, when you create your own application, it *should* be placed completely in this folder or the application can be placed in the hot-deploy or the specialized folder.

base folder : contains java classes , xml files and xml schema files, for OFBiz starting up and configuration.

data folder : contains some files for the database specification.

framework folder : contains the OFBiz framework components , like the Entity Engine, the Service Engine, the common folder that contains files that is common for any application..etc.

hot-deploy folder : this folder can also hold some applications , where the components of these application are loaded automatically without the need for loading them explicitly as we will see later when looking inside an application.

logs folder : the OFBiz uses the log4j project for its logging System, this folder contains the log files.

specialized folder : contains some extra applications like “community” and “wholesale” which are not part of the OFBiz core.

website folder : contains the www.ofbiz.org website html pages.

startofbiz.bat : this file is used to start running the OFBiz.

Now , we would look at an application, so having a look inside the applications folder :

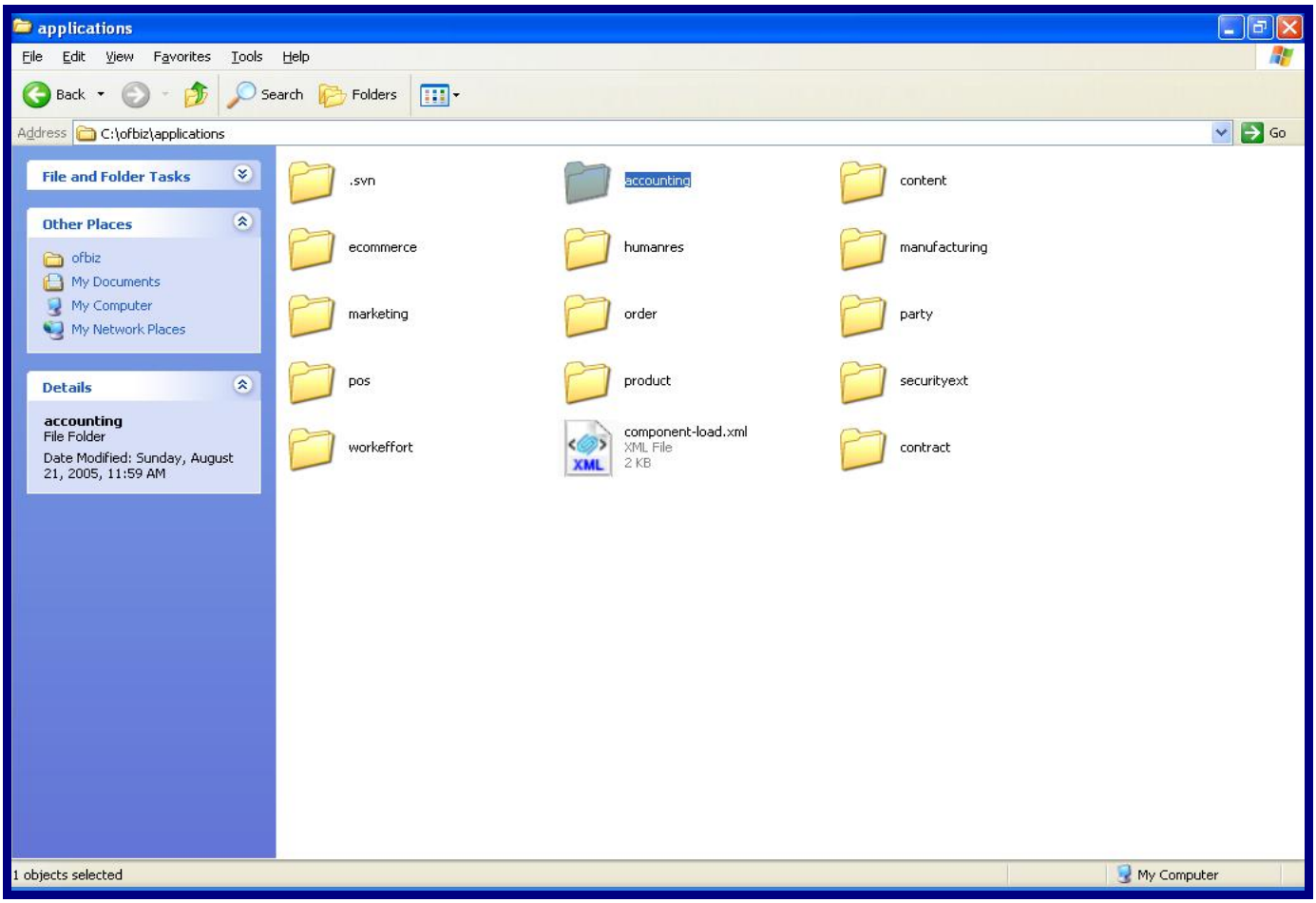


**Figure 4**

here are some applications, the accounting application ,the party application, the the order application, …etc.

The component-load.xml file is a very important file, because without it, the OFBiz can not load any application “unless this application is placed in the hot-deploy folder as mentioned earlier”. Whenever you create a new application, that is you add a new folder beside these other folders “party, order,…etc” , you need to tell the OFBiz to load this application, and this is done with the component-load.xml file. It defines the location for all applications that needs to be loaded when the OFBiz starts.

Here is the load-component file:

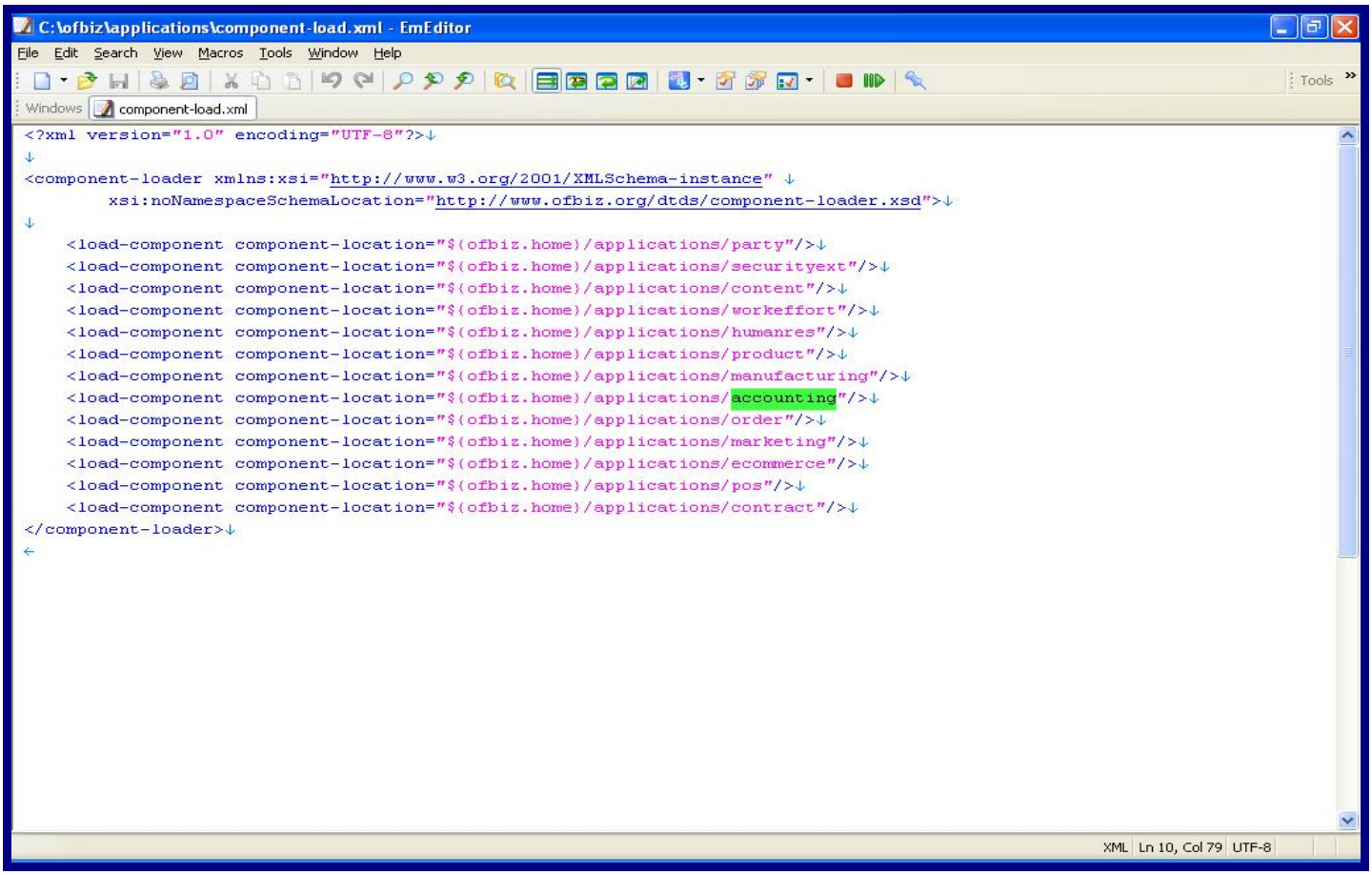


**Figure 5**

*Note : [1]*

*In OFBiz , any application is placed inside a component, that is the OFBiz deals with componets that contain one or more applications ,I guess this is why the file is a "load-component" file not a "load-application" file.*

Now we will have a look at an application, we will take the "Accounting application" as an example , and all the application have the same structure , generally.

## Case Study : Accounting Application :

The accounting application holds many smaller applications inside , one of them is the Agreement . We would go through the three –tiers of this application, i.e., the Data Layer, The Business Logic Layer and the Presentation Layer and the Controller.

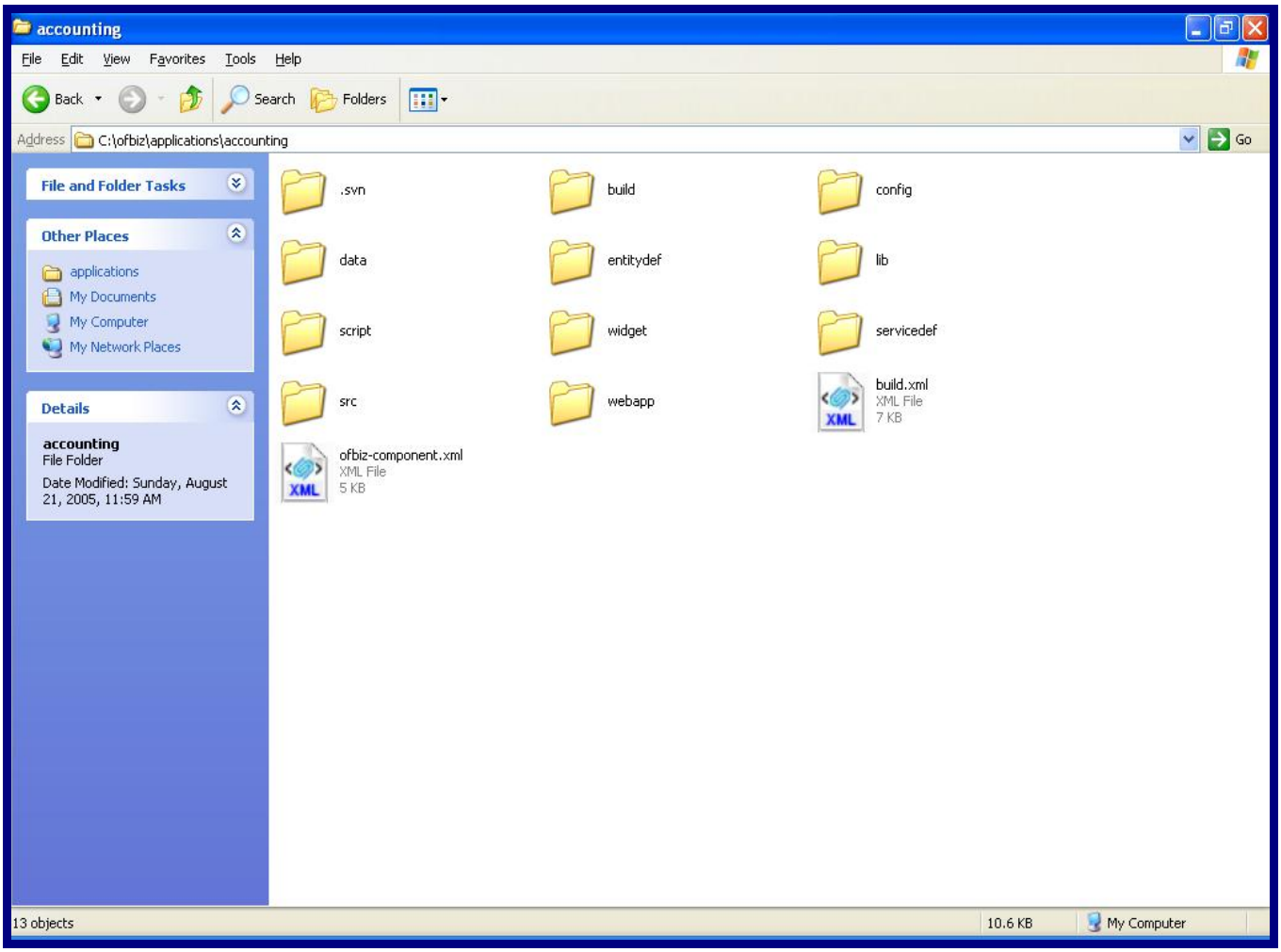


**Figure 6**

These folders are as follows : *[1]*

ofbiz-component.xml : defines this application by specifying where is the location of its
data model : <entity-resource>
business logic : <service-resource>
web applications <webapp …../> .
It is very important to notice that any entity resource file or service resource file should be referenced to in the ofbiz-component.xml.

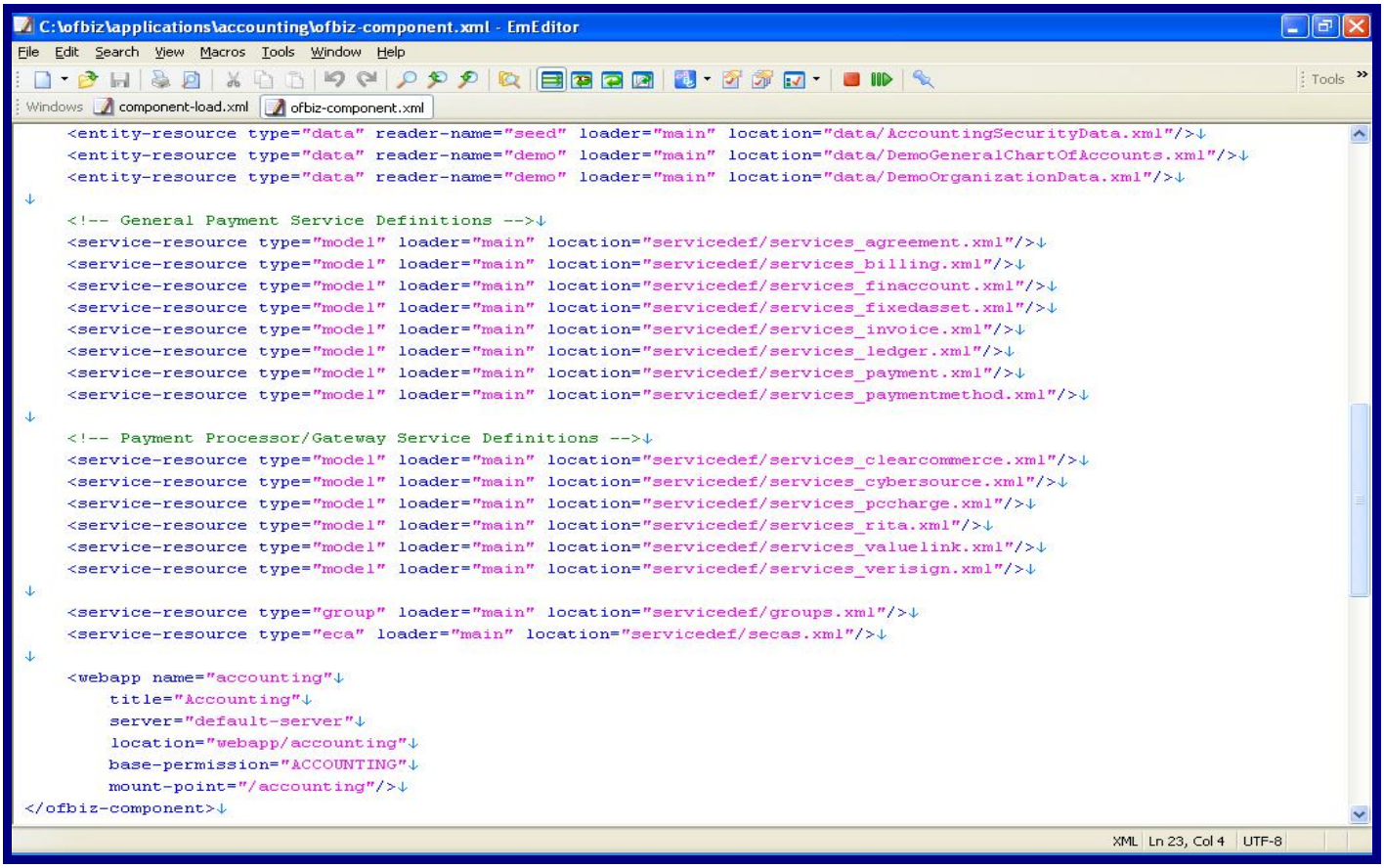


**Figure 7**

build.xml : as its name states, this file is used to tell the ant program how to build the OFBiz application.

.svn : contains the weekly updates batches to this application.

build : contains the java compiled codes ".class files " and the libraries for the accounting application.

config : used generally  for data configuration , an example is , it is used to support different languages interfaces , inside it you will find some files for different languages, and based on the user interface language, one of these files will be used.

data : contains the seed data “data loaded when ofbiz starts” and the demo data.

*Finally, we are last with the entitydef and the servicedef. For these two we always have two parts : definition and implementation.*

entitydef :

contains the data layer definitions and implementations, i.e. the database relational tables and their relationships.
Inside this folder , there will always be two main files, one for definition and the other for implementation.

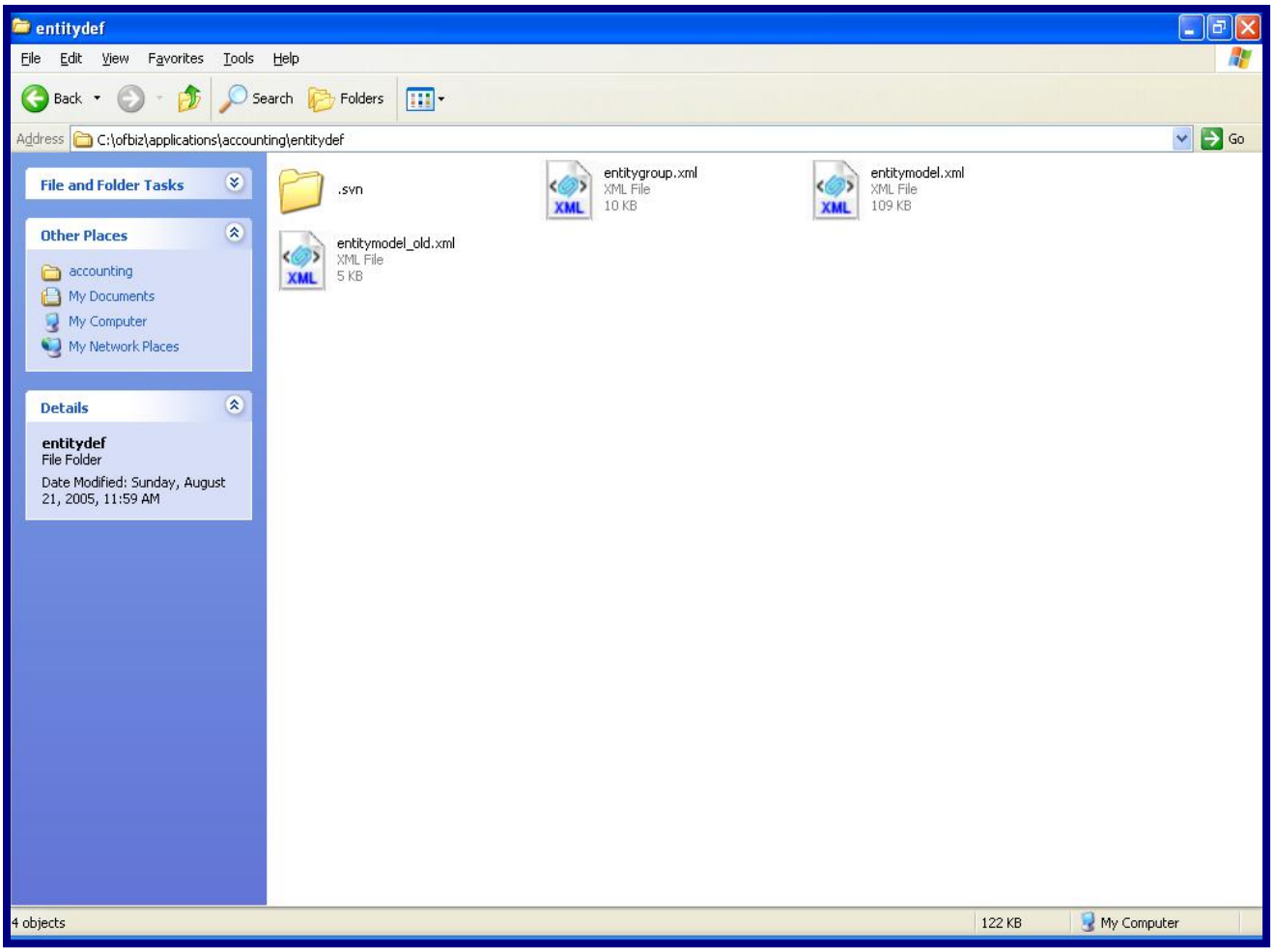


**Figure 8**

**entitygroup.xml :**

It defines the tables of the database. For example, we have Agreement , AgreementAttribue,AgreementItem,AgreementItemAttribute,….etc .
It is strange as we are studying the Agreement applitcaion, which is under the accounting application , but its definition is located in the entitydef directory of the party application!. May be it will be moved to the accounting application soon!. Nevertheless, all the applications follow the same pattern, and even if the Agreement Entities were taken out from the pary Application and replaced in the accounting Application it would work exactly the same.

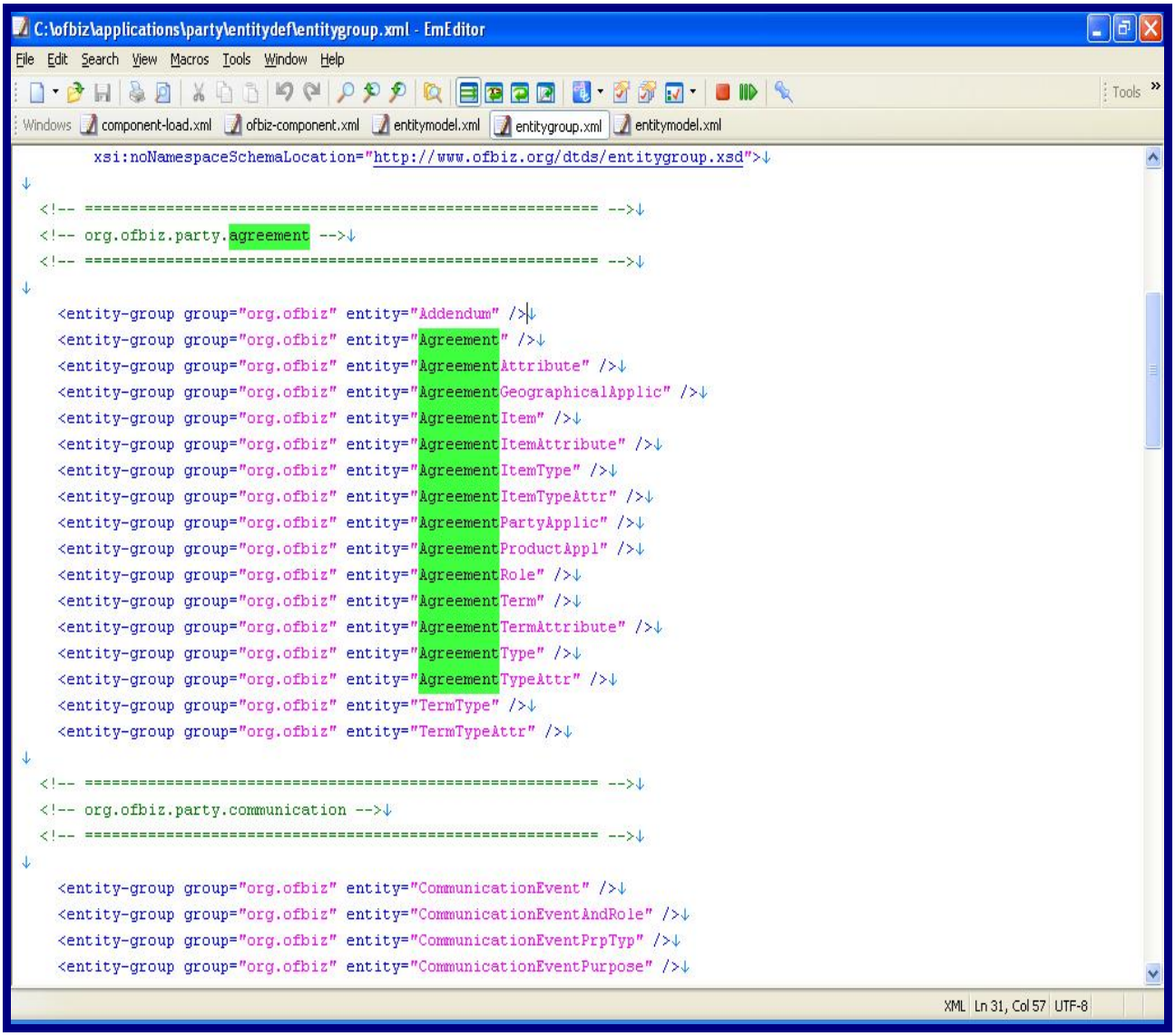


**Figure 9**

**entitymodel.xml :**

It implements these tables that were just defined in the entitygroup.xml, i.e. , it gives details about their fields, types, relationships, …etc.As an Example, the Agreement table or Agreement entity:

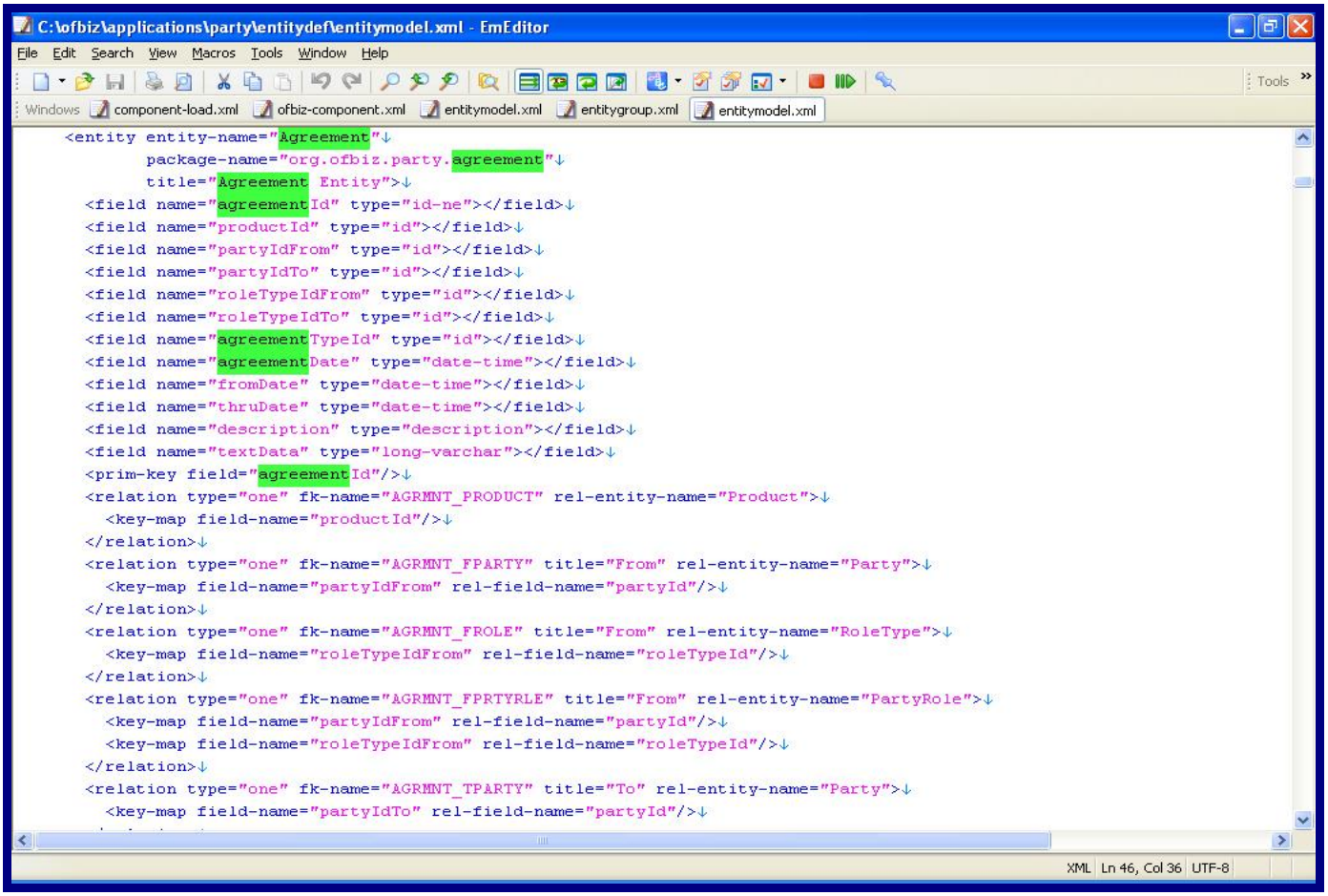


**Figure 10**

Note that each filed has a field-type. Field types might differ based on the type of the database. Thus, based on the database you are using “the default for OFBiz is the Derby database” you would decide what types to choose for your fields.

To know the different types for the database, you could follow the directory :

*C:\ofbiz\framework\entity\fieldtype*

Inside , you would see many files, each for a particular database.

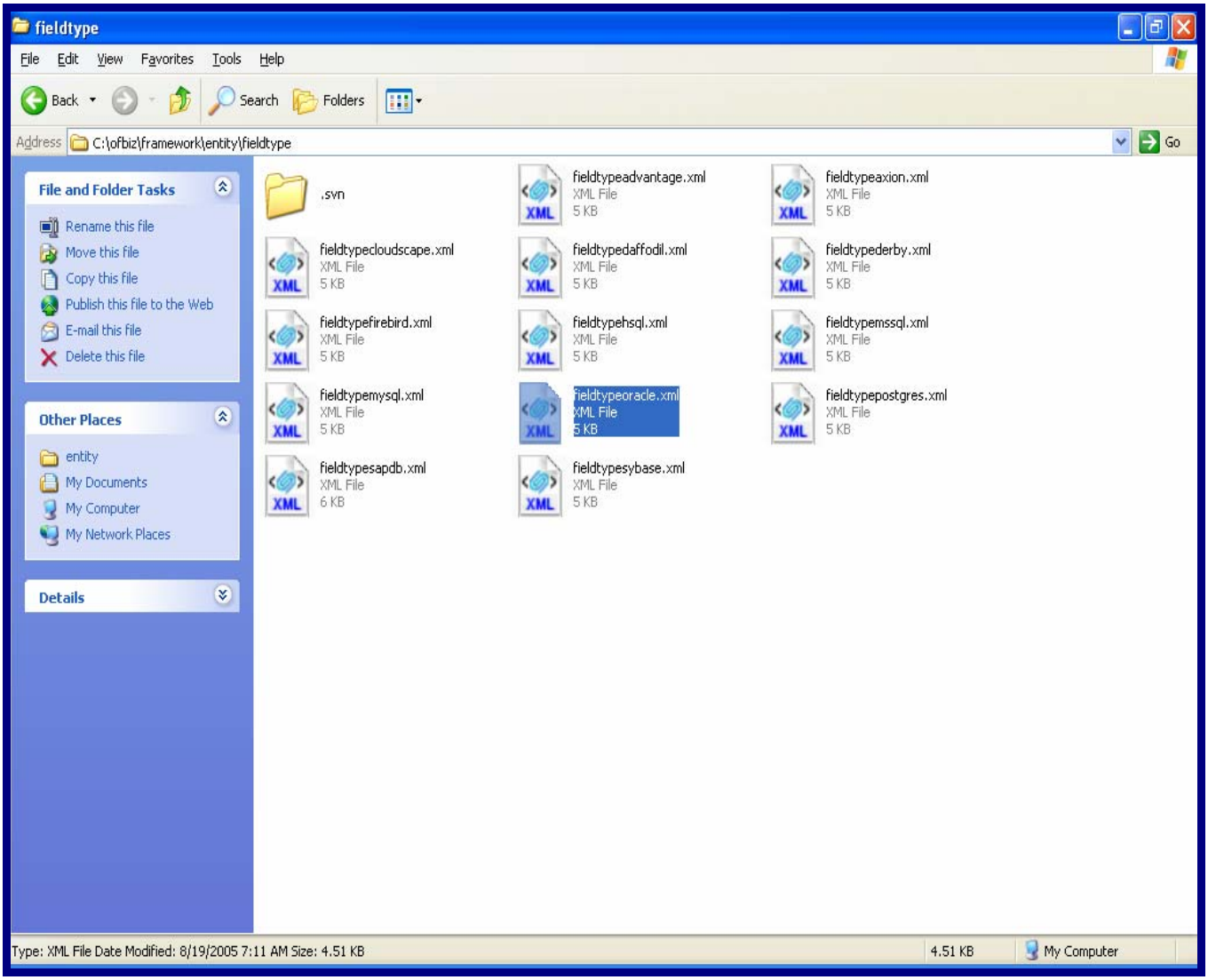


**Figure 11**

Assuming you are using the Oracle database , you would check the fieldtypeoracle.xml file , as shown below.

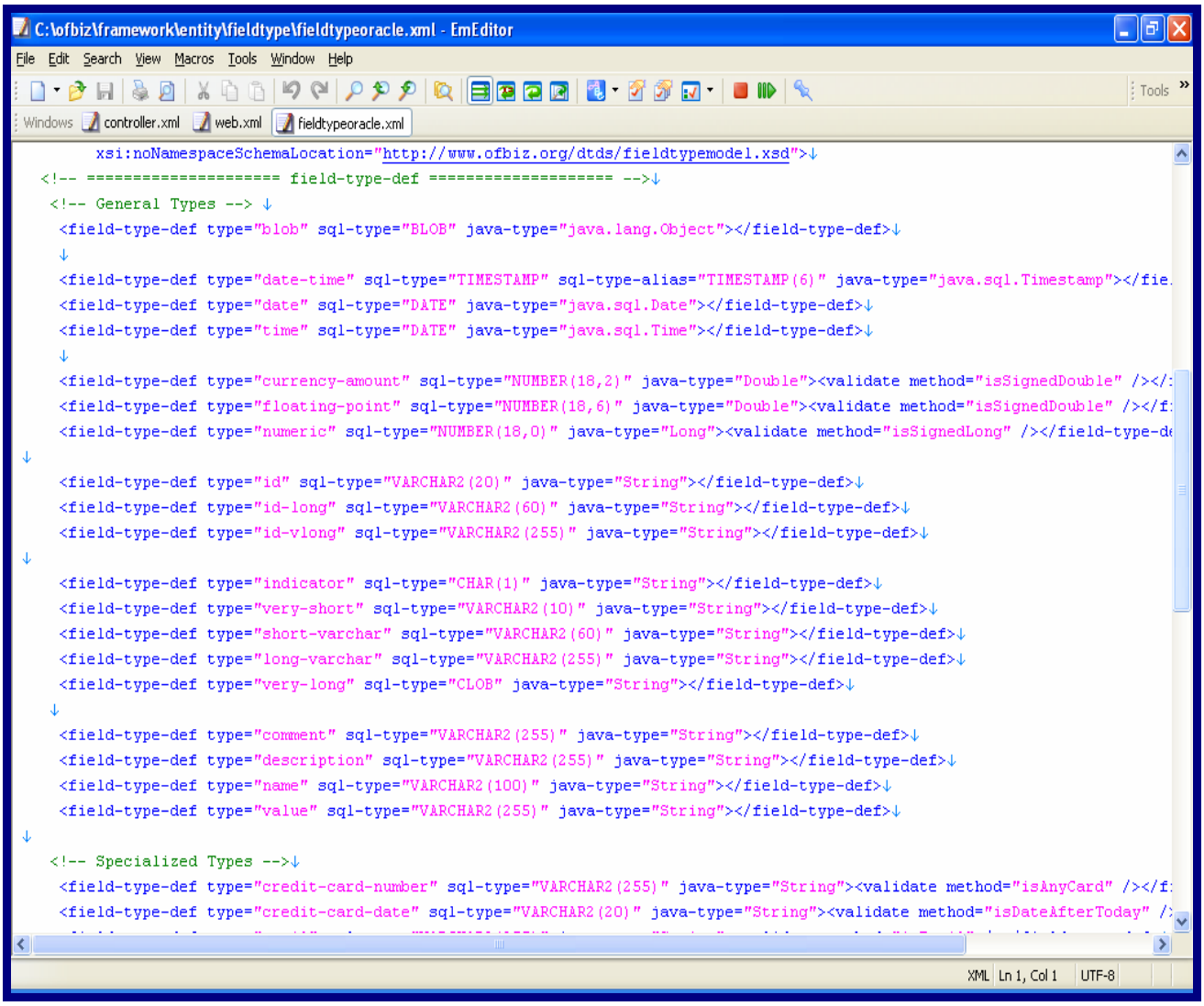


**Figure 12**

Notice that you are not restricted to these type, you can add your own ones, all what you need to do is to add a new <field-type-def> tag.

For more information about the entity model and entity definition, you can visit : Entity Model .

servicedef :

It defines the services used in the business “logic” layer, it contains the services.xml file , which define the services. In our case , the Accounting application has many sub-applications in it, one of which is the Agreement as mentioned earlier. Thus, the services file is services_agreement.xml

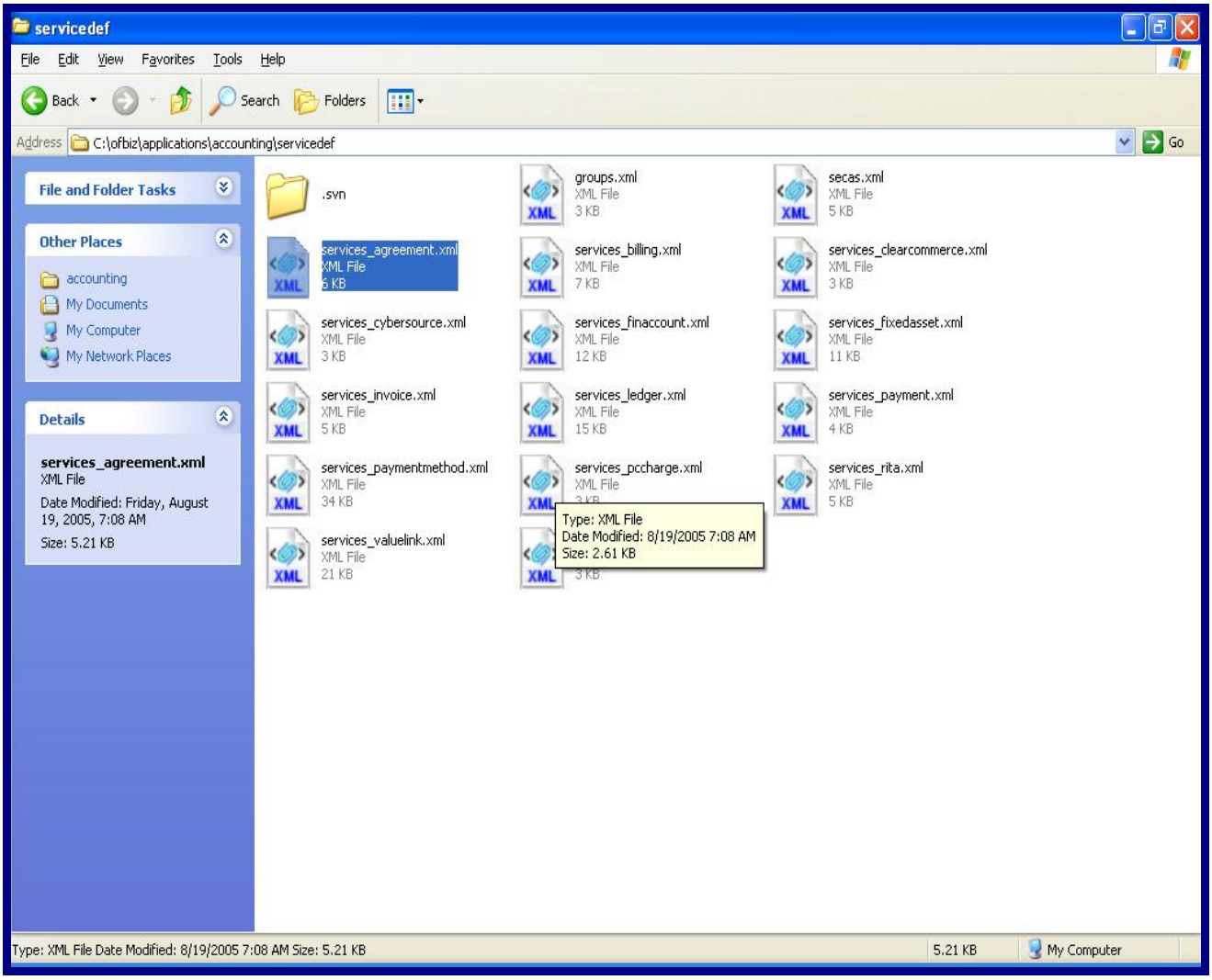


**Figure 13**

***Important Note :***

Whenever you add a new service file , like the service_agreement.xml or any service definition file, you need to include a reference to it in the ofbiz-component.xml file “Figure 7”

Having a look inside this file, service_agreement.xml , we would see the definition of all the services used by the agreement application.

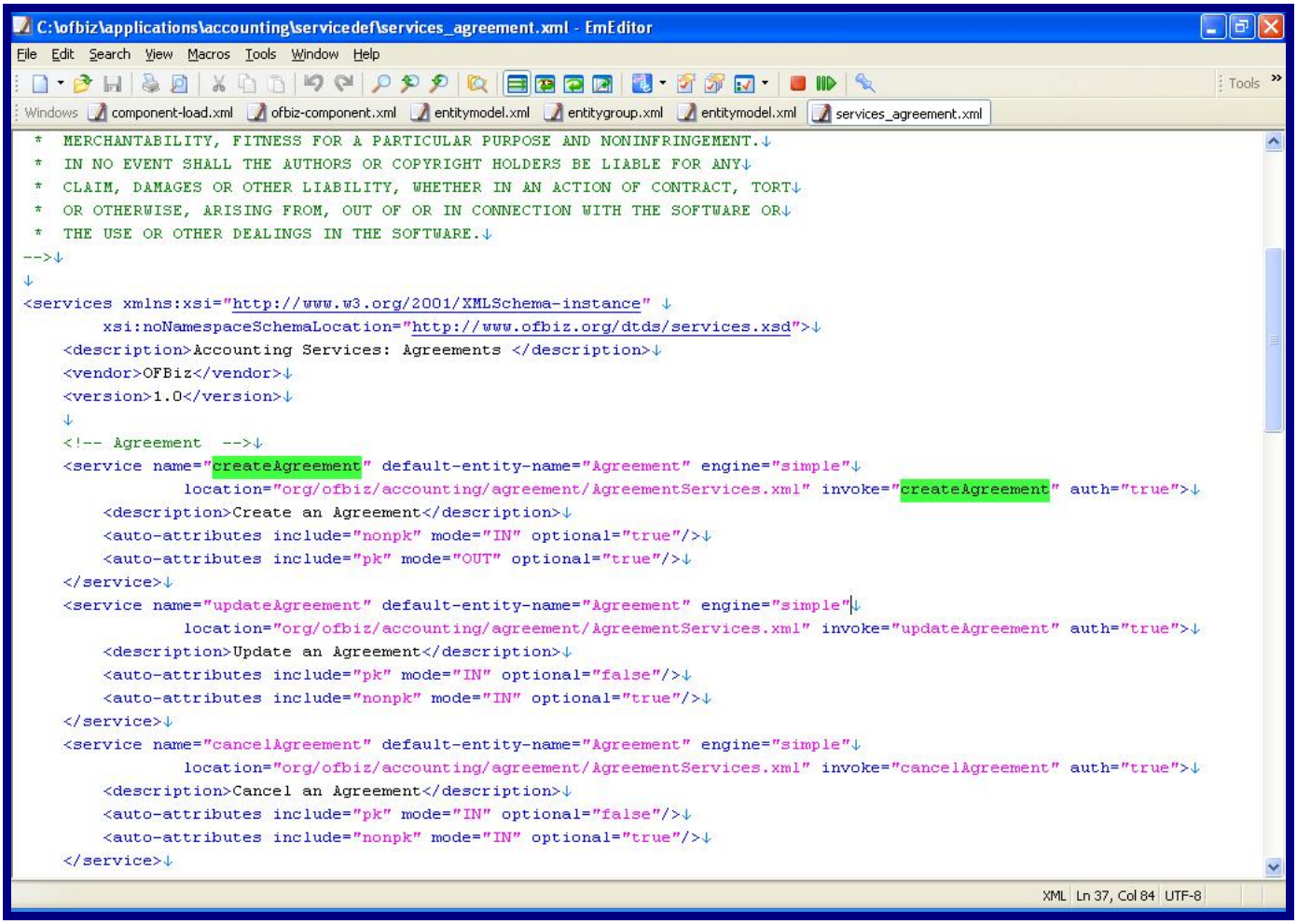


**Figure 14**

After defining the services, we need to implement them. Normally, services are implemented using the OFBiz mini-language. However, if the service cannot be implemented with “xml” , we can use java to implement it.

script : contains the implementation for the services using the OFBiz mini-language, and it contains some scripts.
Inside this folder, we will find many subfolders containing the service implementation for the different accounting subapplications. We are interested in the Agreement subfolder :

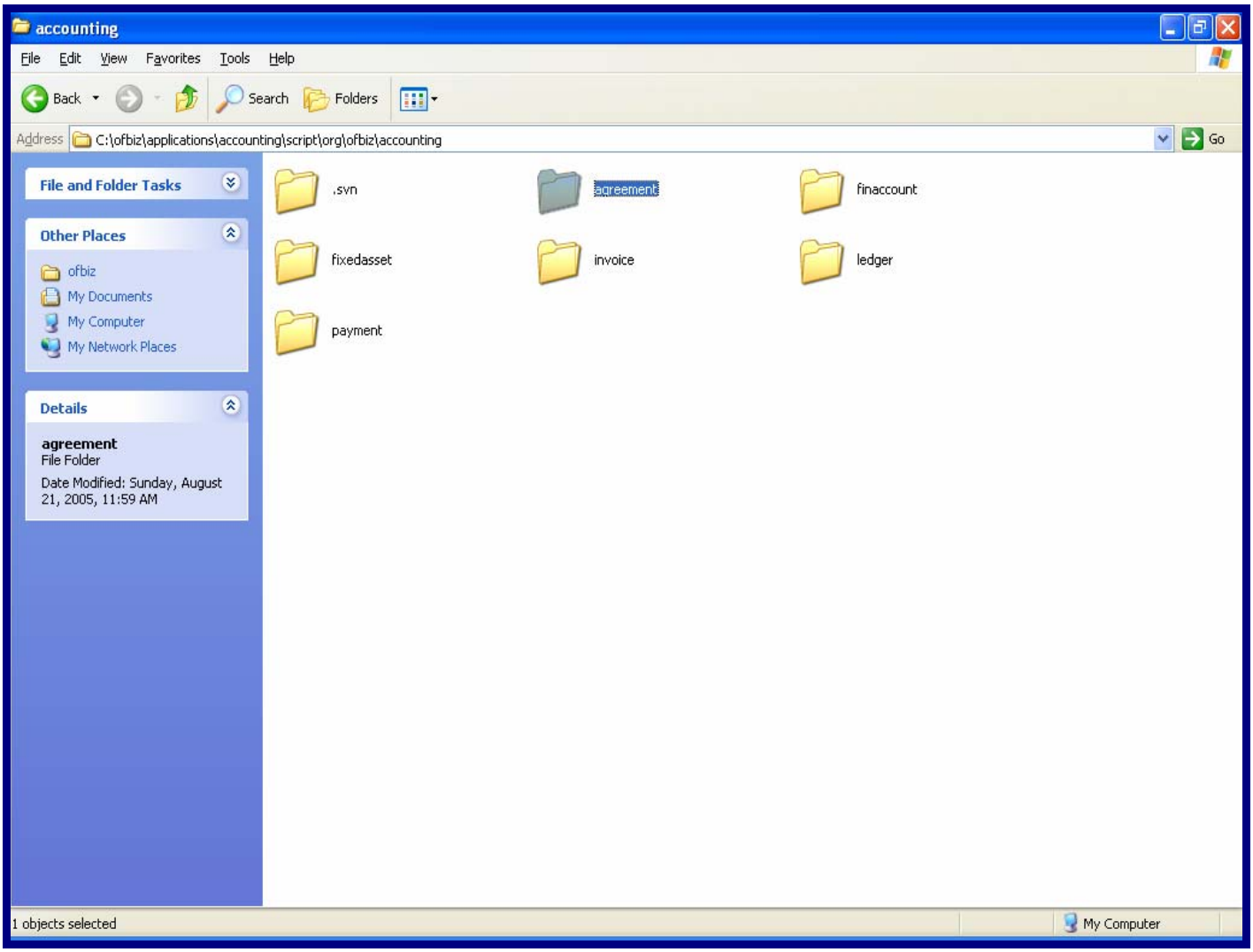


**Figure 15**

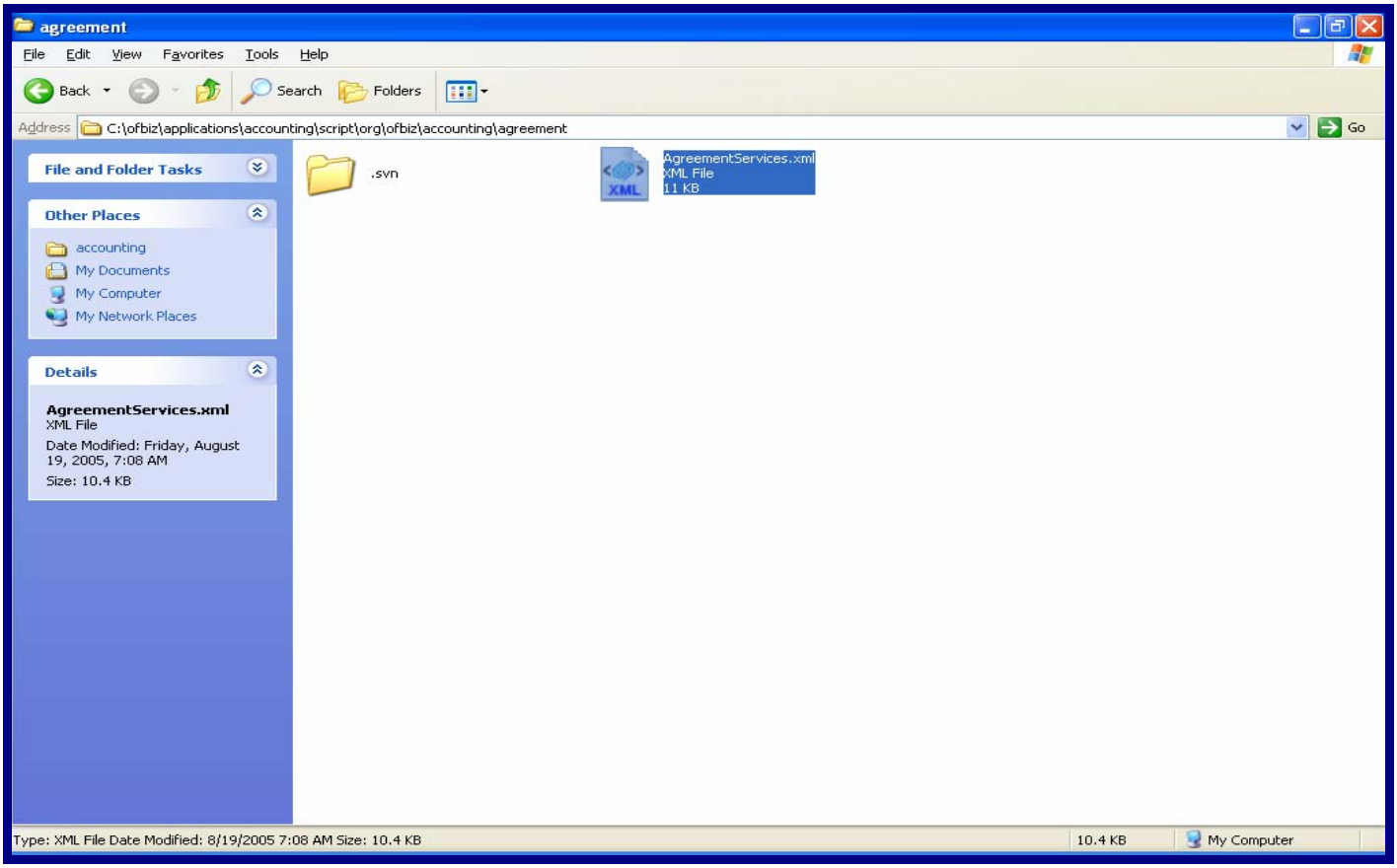


**Figure 16**

and having a look into this file :

for example. It is implementing the createAgreement service that was defined in the servicedef , as was shown in Figure 14.

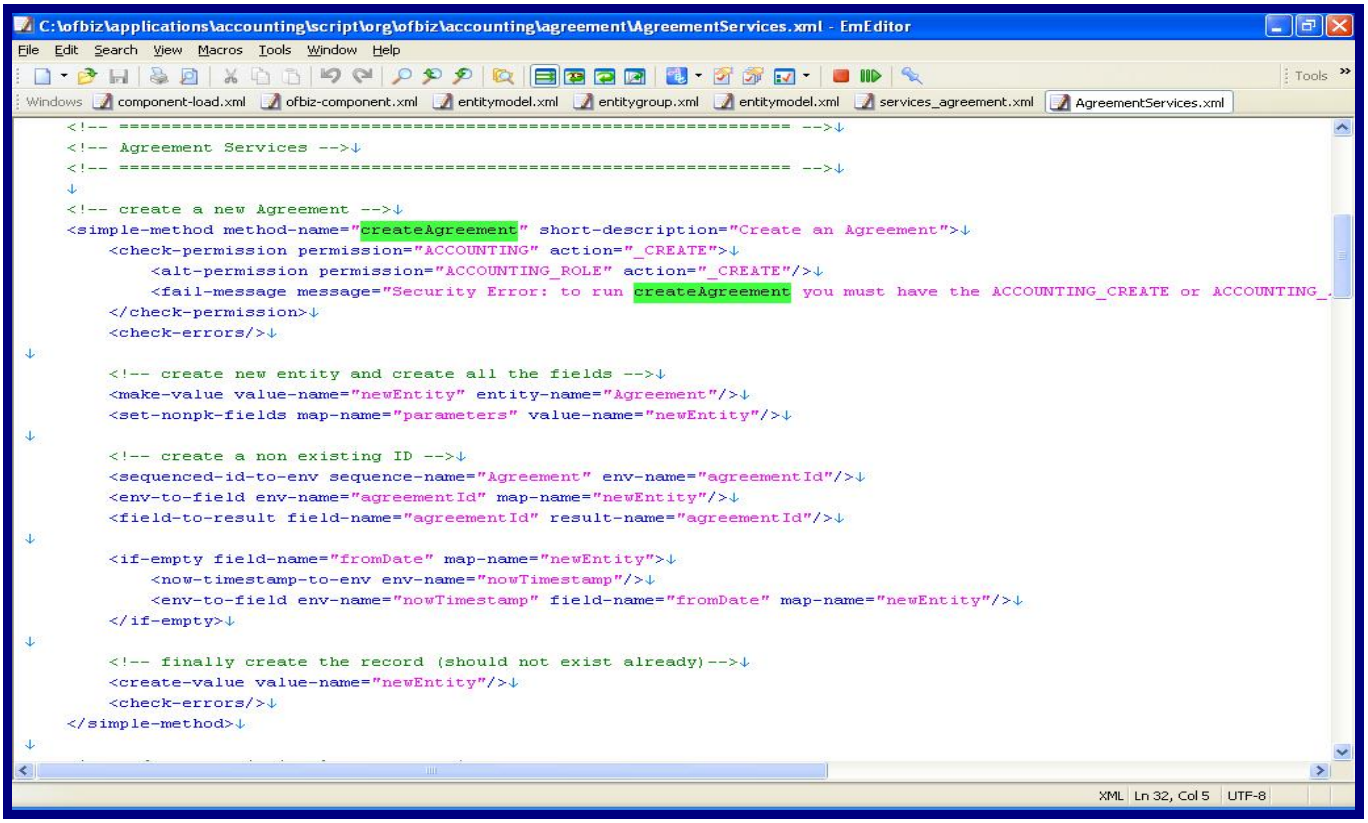


**Figure 17**

For more Information on how to define services , you can visit :
Service Engine Guide.

src : contains the java source files for the services that were implemented with java.

widget : recently the OFBiz presentation layer pages are defined as "Screens".This directory holds "widgets" for the user interface screens. OFBiz allows the user interface design to be created as "generic screens" rather than just web pages, so they could be reused eventually for some other platforms. The widgets/ directory's contents mirror those of the webapp [1]* So, each application will have its own screens , as so the Agreement application does. Inside this folder, we would find the AgreementScreens.xml file that defines the Agreement screens.

In the Figure below, Figure 18 , we would see the AgreementScreens.xml file among the other applications screens files.

In Figure 19, we will see the findAgreementScreen that allows to search for a particular agreement.

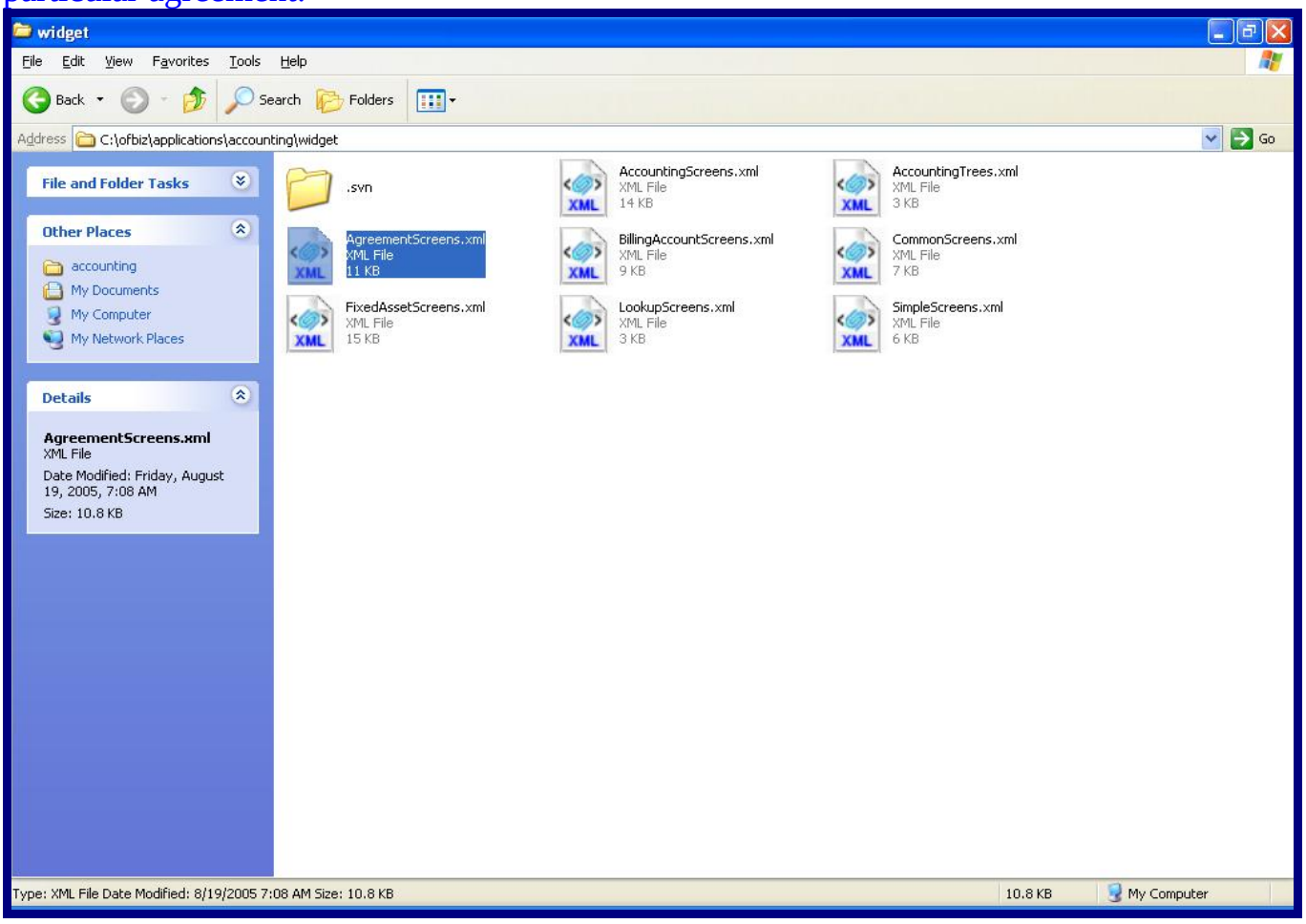


**Figure 18**

Screens are divided into two parts : actions and widgets.Actions are responsible for data retrieval while widgets are responsible for data display.

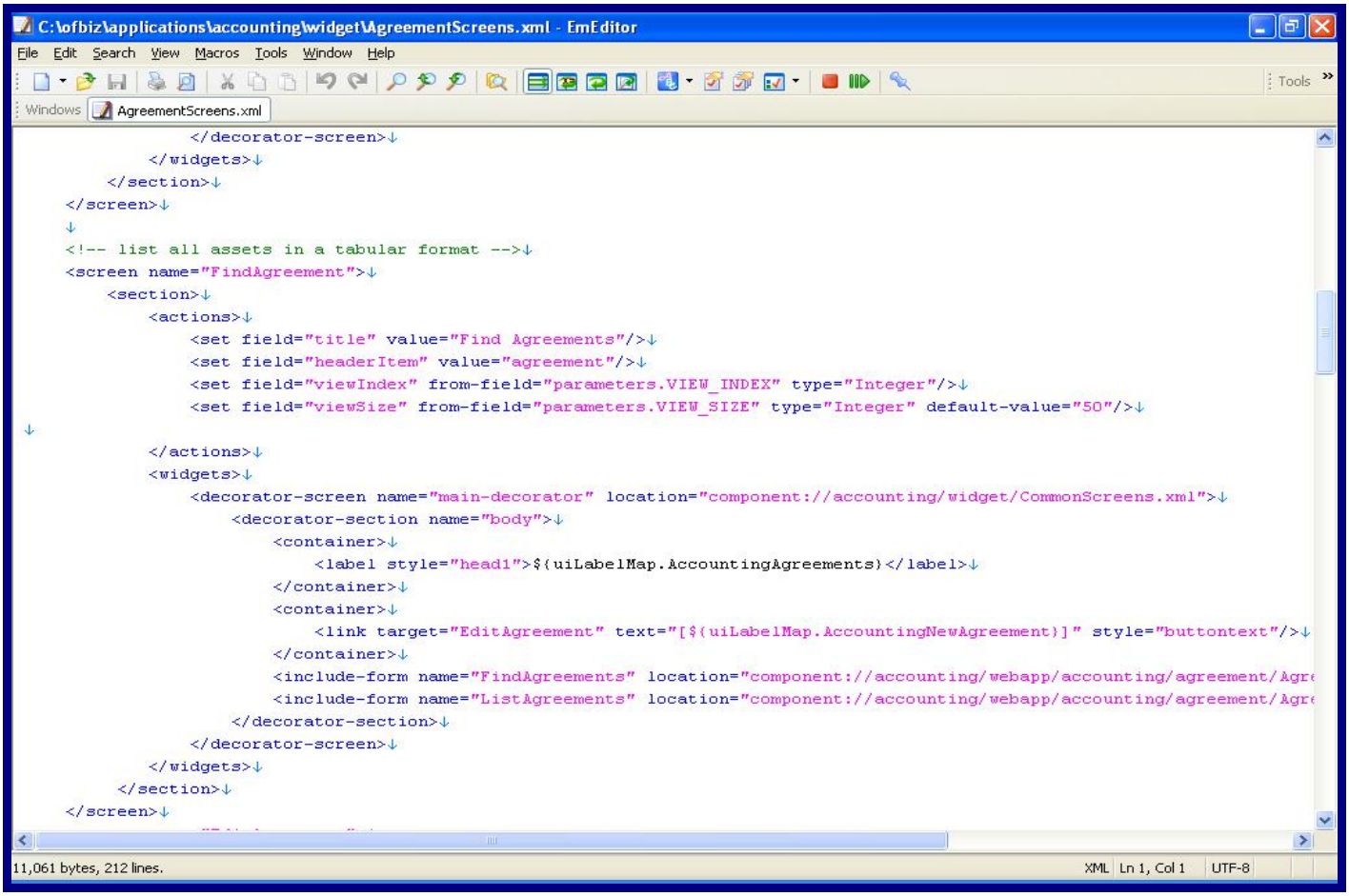


**Figure 19**

webapp : contains web application pages and forms . With OFBiz, pages are divided into smaller pieces which are re-combined to create the final product. Thus, many pages can share common elements such as page headers, sidebars, and navigation bars. This is called the "decorator pattern." There is a further separation of the activities of a page into "actions," such as getting data from a database, and "presentation," the display of that data to the visitor. ***[1]****

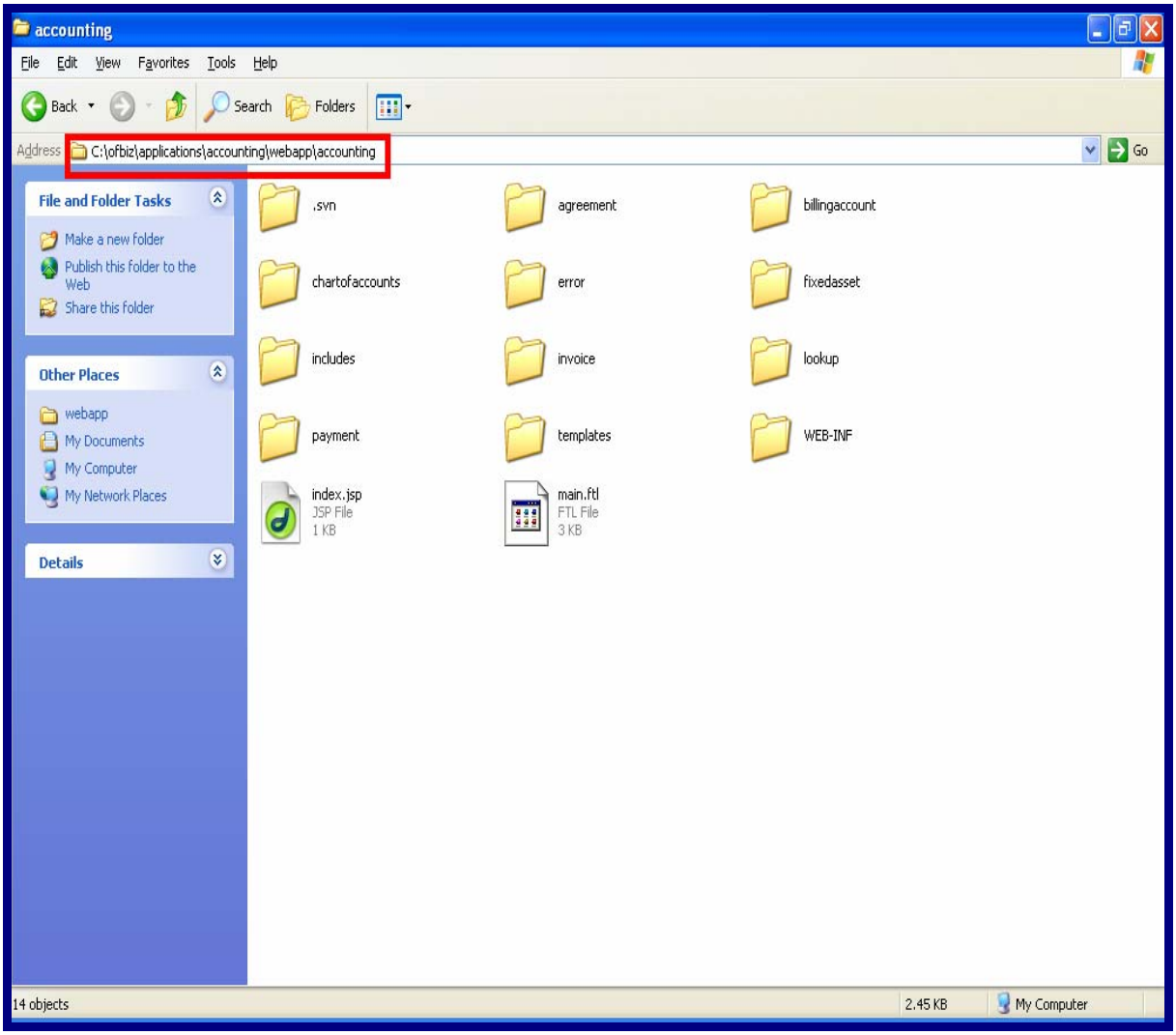


**Figure 20**

The basics files/folders in our application are :

Index.jsp : used to redirect the controller to the main page.

main.ftl : The main page for the accounting application, written with FreeMarker Template Language (FTL) .

includes folder : contains the appheader.ftl file that is common for all the accounting application .It can also contain some other ftl files if needed to be used by the application.

error folder : contains the error pages to be displayed when a particular error occurs.

Agreement folder : contains the agreement forms that are used /called by the agreement screens or agreement ftl files. Here is the findAgreement form as an example

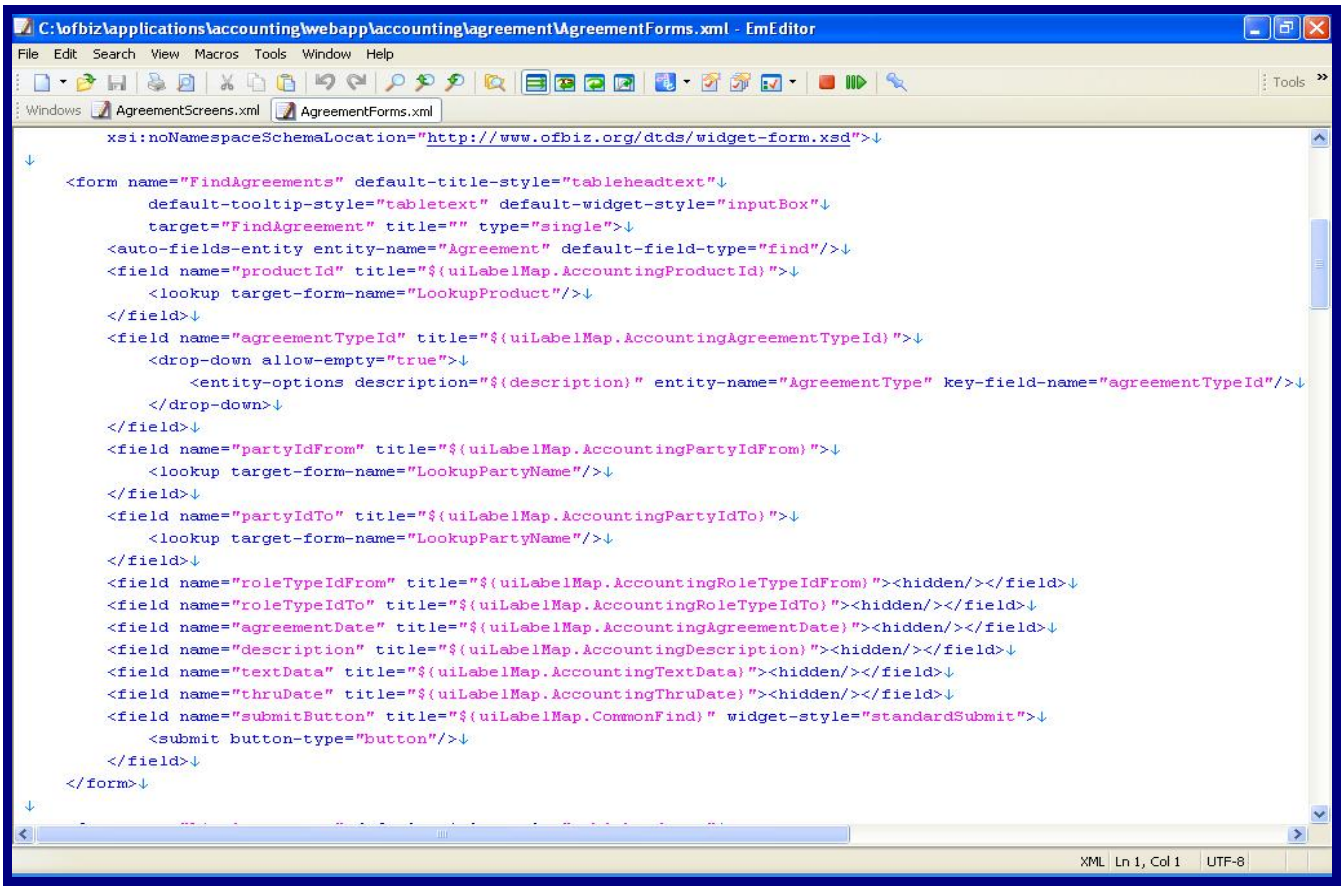


**Figure 21**

WEB-INF : the most important directory , it contains very important files and directories.

-actions folder : has beanshell scripts that are used to process and gather data from the database .

-web.xml file : discussed earlier.

-controller.xml : Resposible for controlling the coming request .Any request to the application, wither it is a screen request, service request, event..etc, it should be passed through the controller.

## Inside the controller.xml file :

## 1) Defining the different handlers for different types of events.

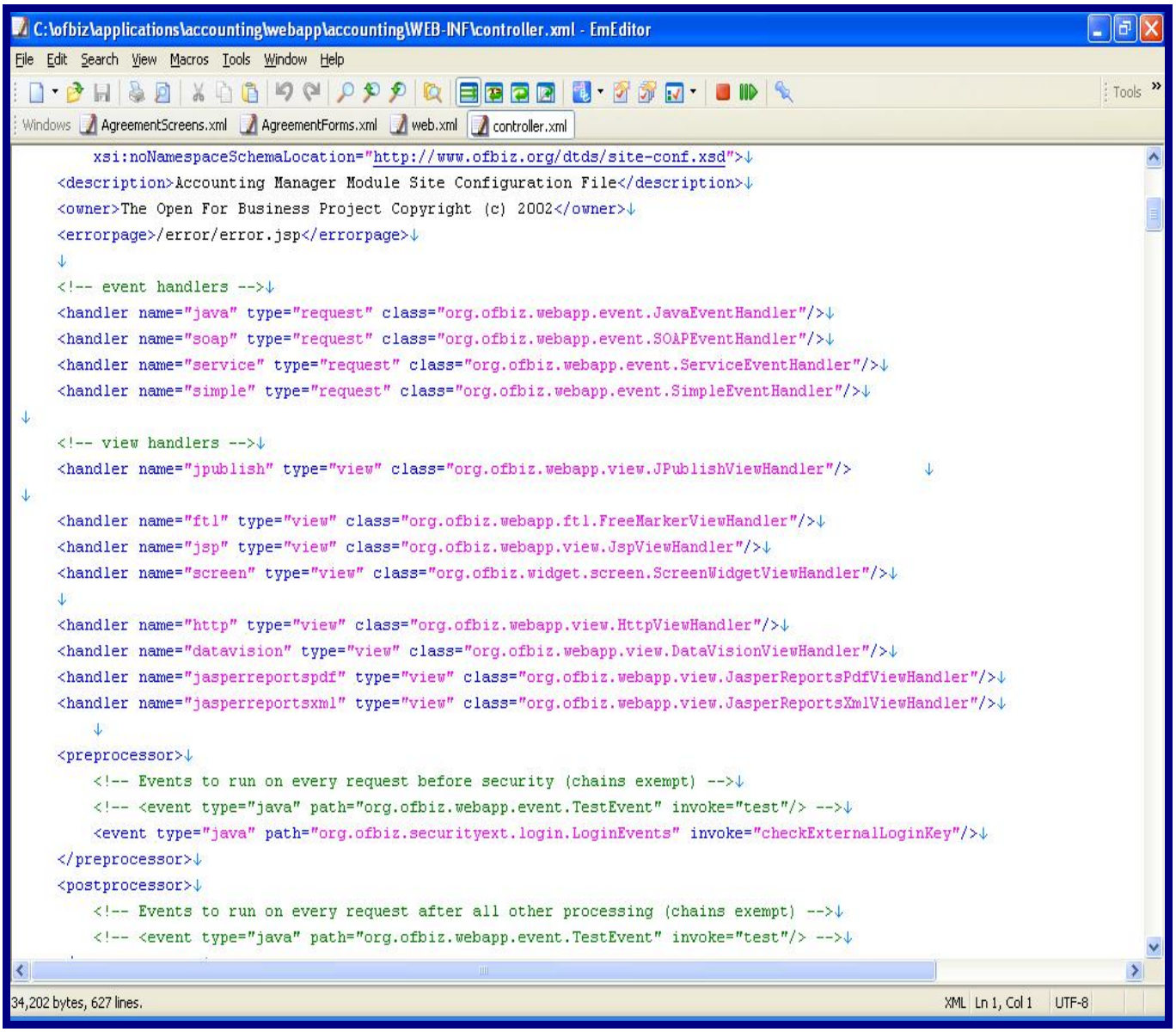


**Figure 22**

2) Defining the request mapping for the application, it can be a screen “view” request or a service request , as example : FindAgreement and createAgreement.

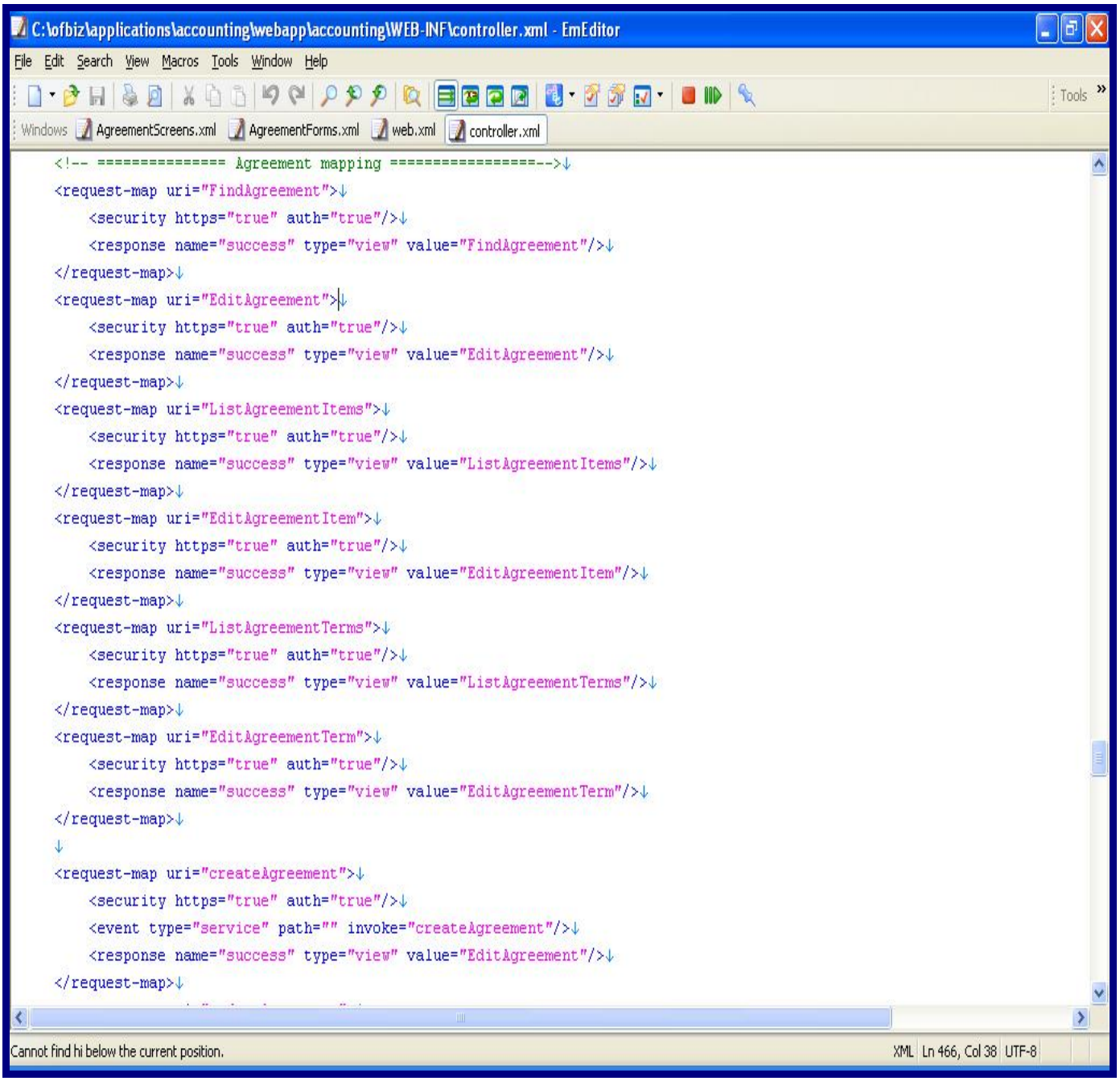


**Figure 23**

## 3) The controller tells where to look for the requested screen or service .

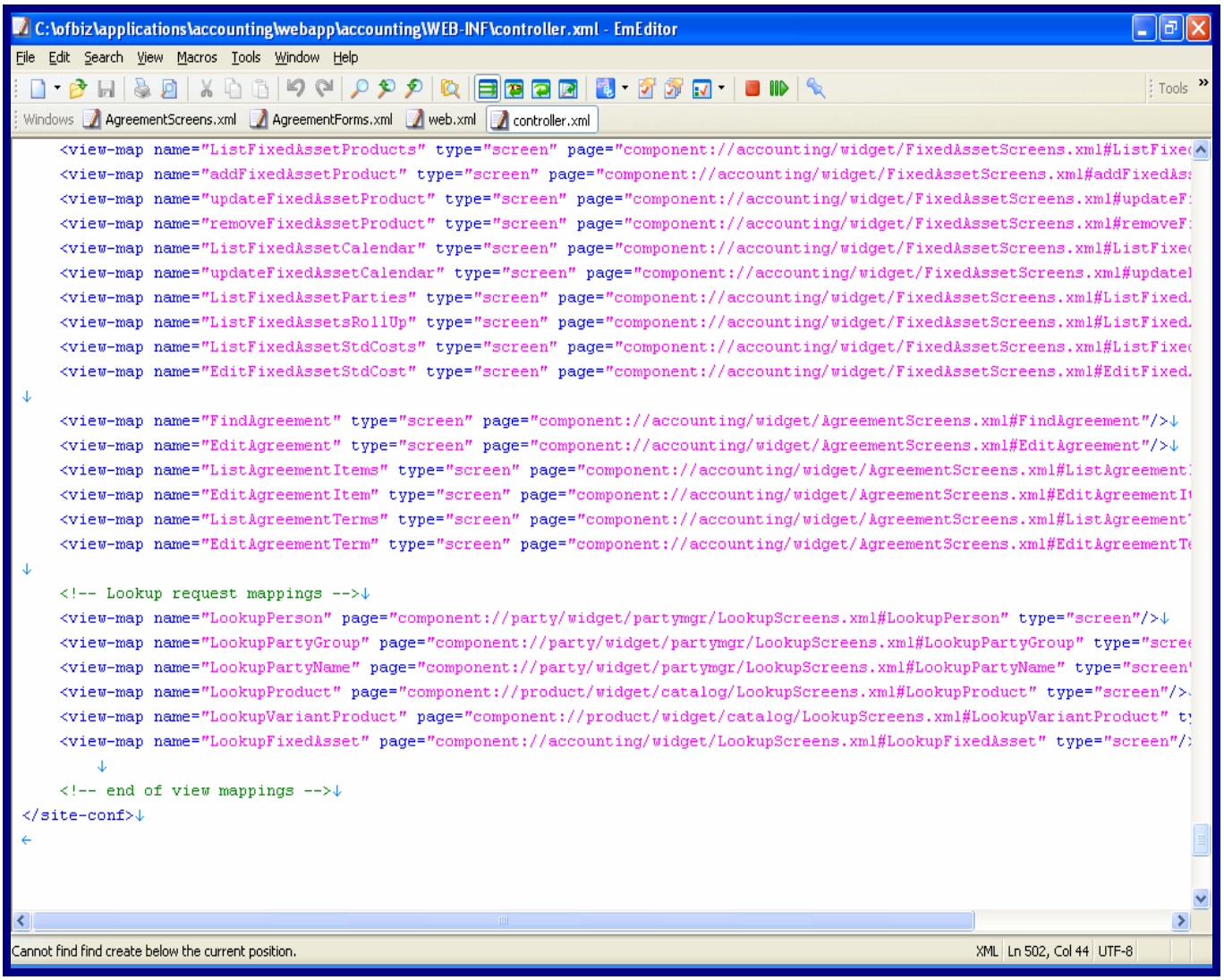


**Figure 24**

**Demo :**

1) Double click on the startofbiz.bat file in the directory C:/ofbiz or :

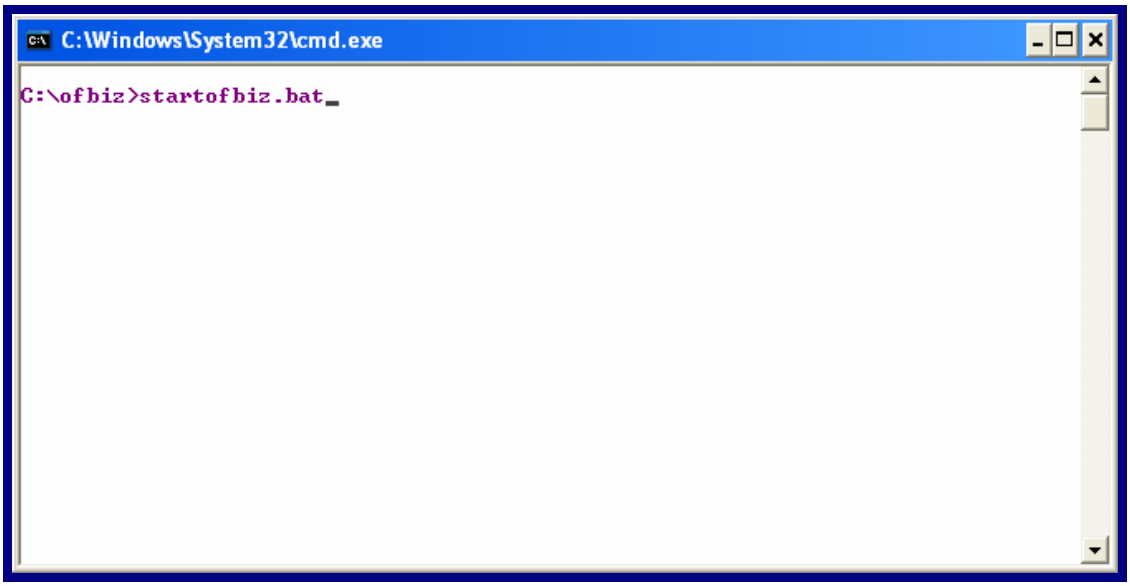


**Figure 25**

2) Wait until the OFBiz runs fully, then in the browser type the following :
https://localhost:8443/accounting/control/main

3) A user name and a password are required : the defaults are
User : admin
Password : ofbiz

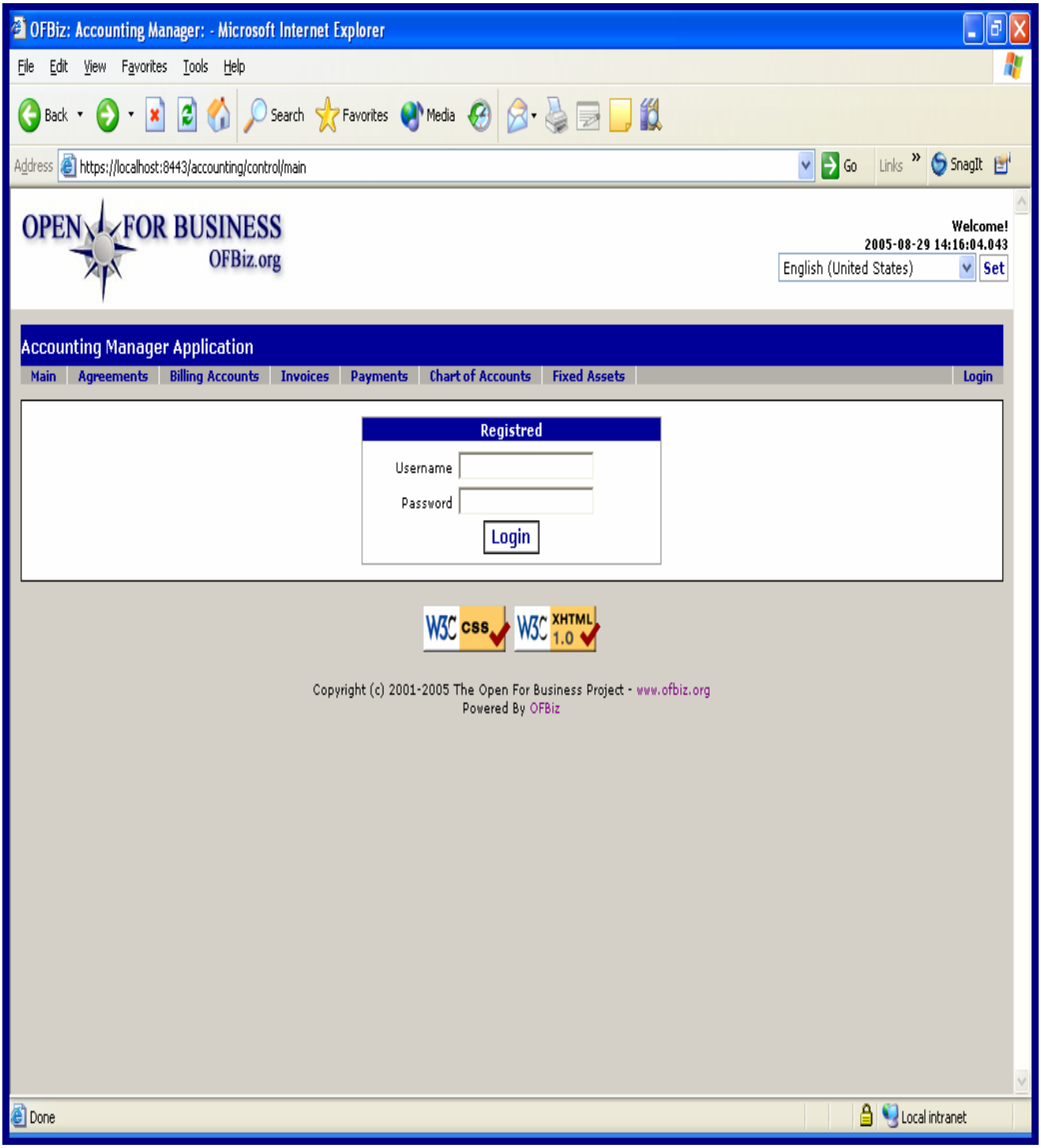
OFBiz: Accounting Manager: - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address https://localhost:8443/accounting/control/main
Go Links SnagIt
OPEN FOR BUSINESS
OFBiz.org
Welcome!
2005-08-29 14:16:04.043
English (United States) Set
Accounting Manager Application
Main Agreements Billing Accounts Invoices Payments Chart of Accounts Fixed Assets Login
Registred
Username
Password
Login
W3C CSS
W3C XHTML 1.0
Copyright (c) 2001-2005 The Open For Business Project - www.ofbiz.org
Powered By OFBiz
Done
Local intranet

**Figure 26**

Now we are inside the application , the first page will be the main page .
It is obvious! The requested page is the main page, and it is requested from the controller.

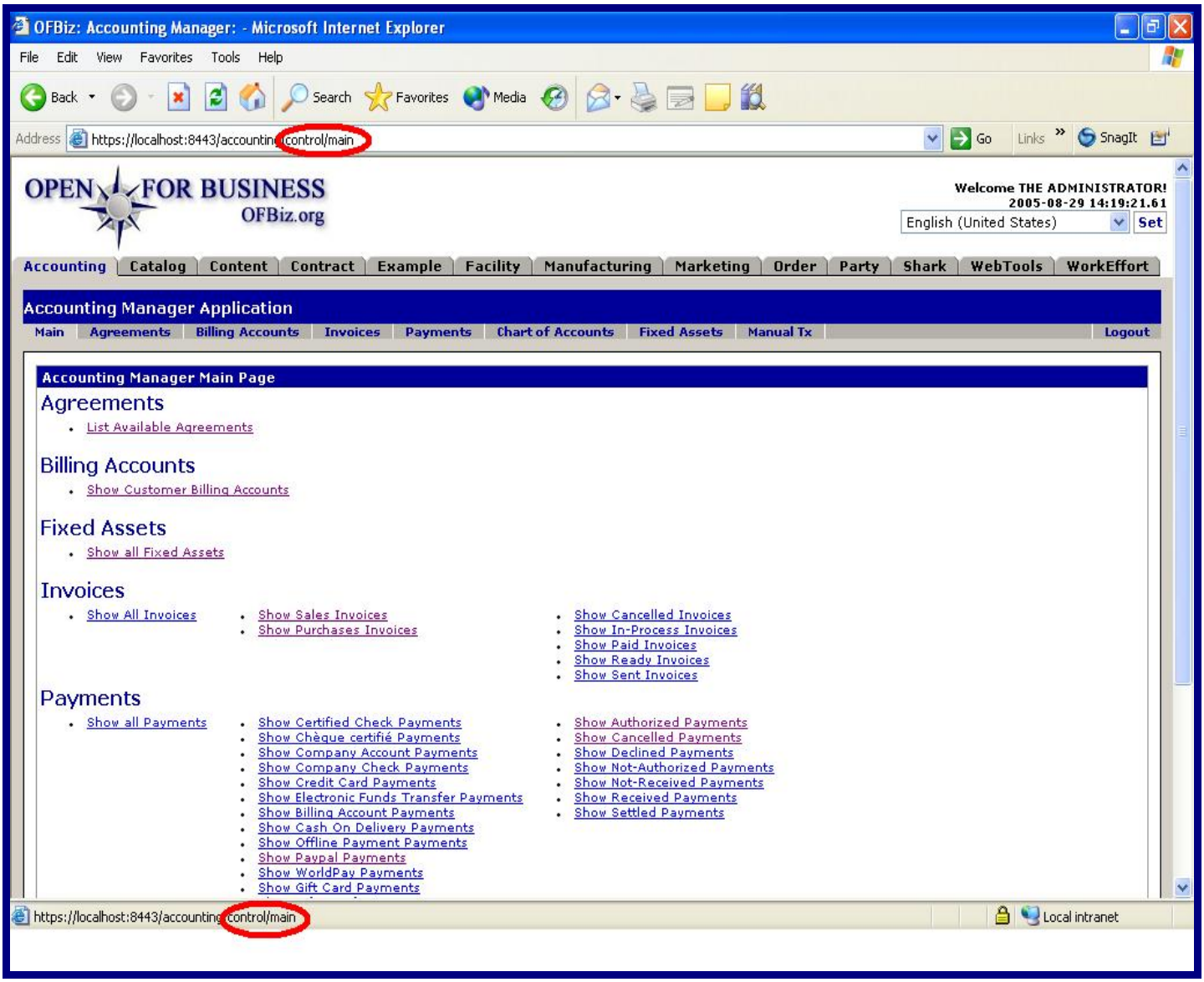


**Figure 27**

Now we will look for the “main” in the controller.xml file :

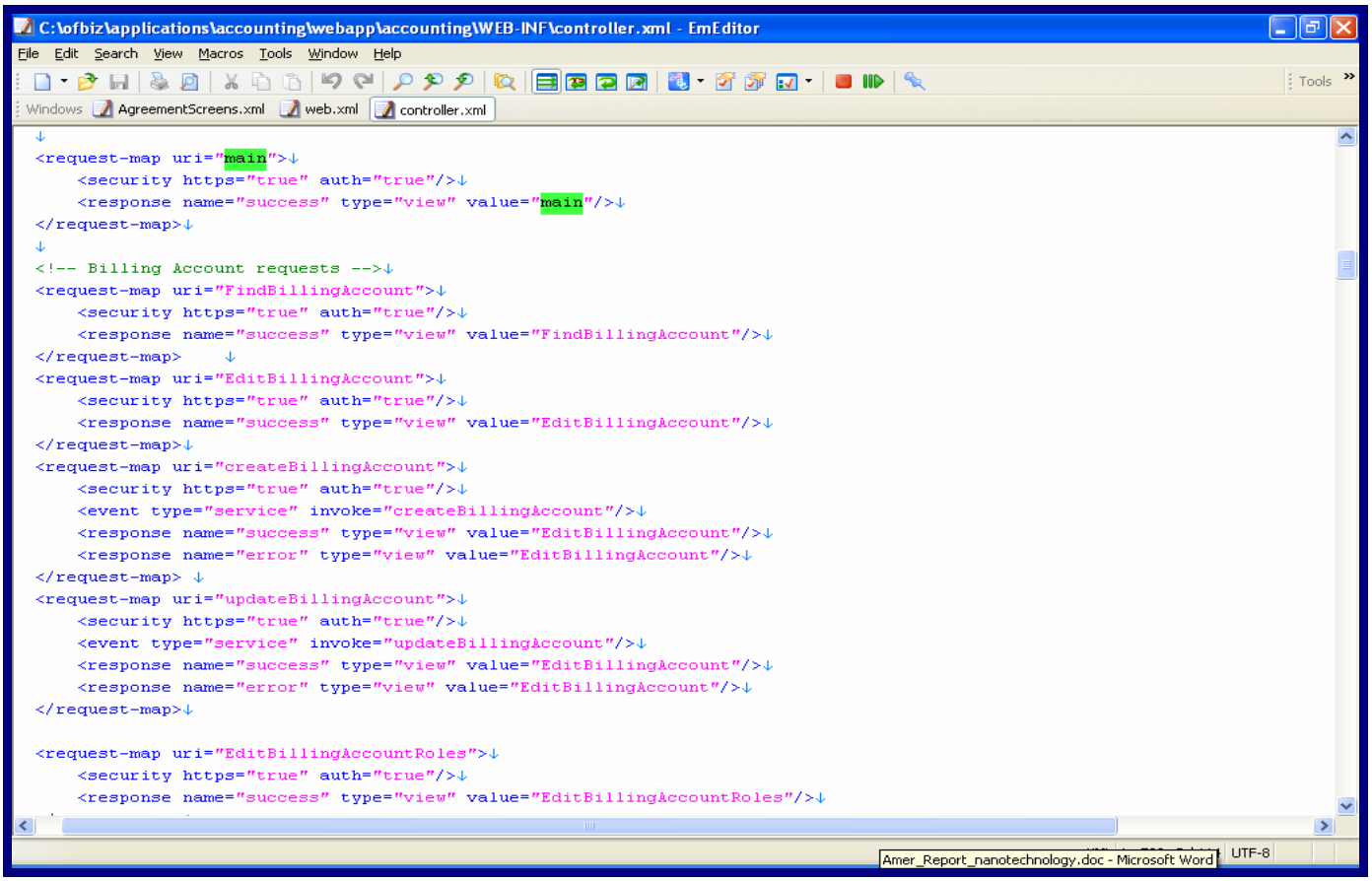


**Figure 28**

the requested map is “main” and when success , this “main” is of type “view” which means it is a screen, not a service , and its value is “main” .

Then we will search for this view at the end of the controller.xml file, whose value is “main”

we will find :

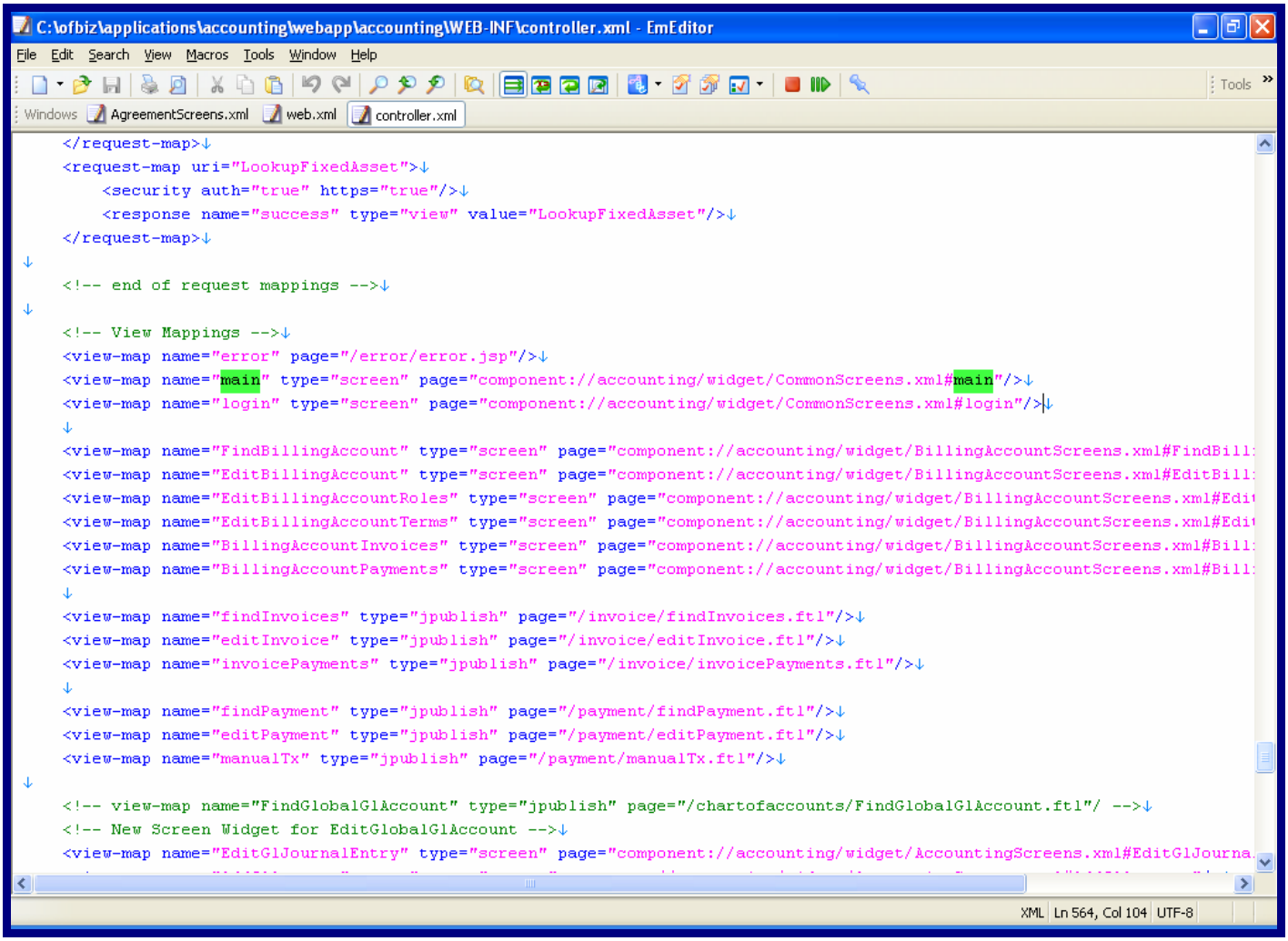


**Figure 29**

Here it gives the path for the screen. We should follow the path in the page=”….. . There is a CommonScreen.xml file and inside this file there is a screen called “main” . “the name after the # symbol is the screen name”.

Now we will follow this screen, we will go to the widget directory inside the accounting application :

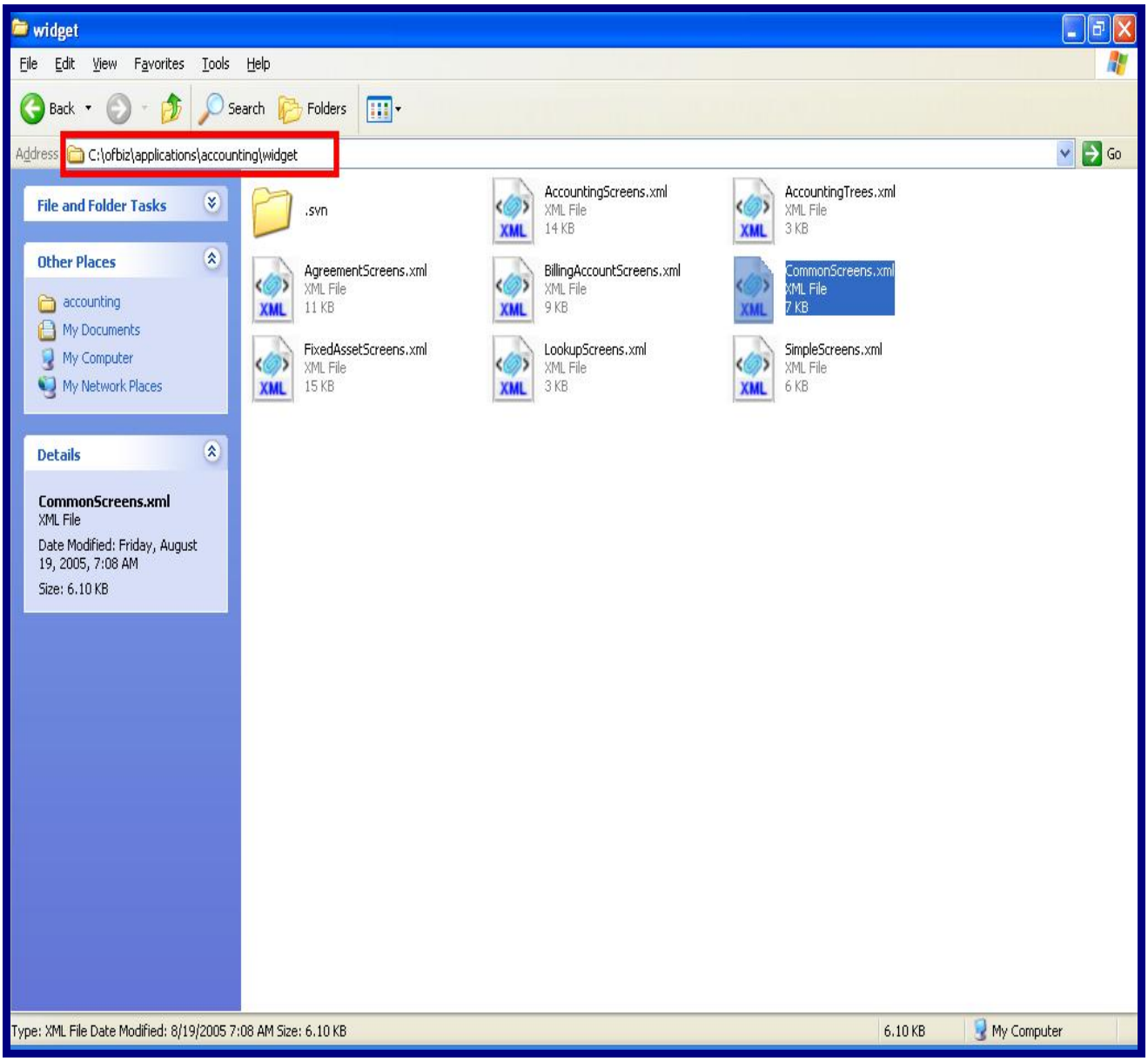


**Figure 30**

inside the CommonScreen.xml file, we will look for the “main” screen.

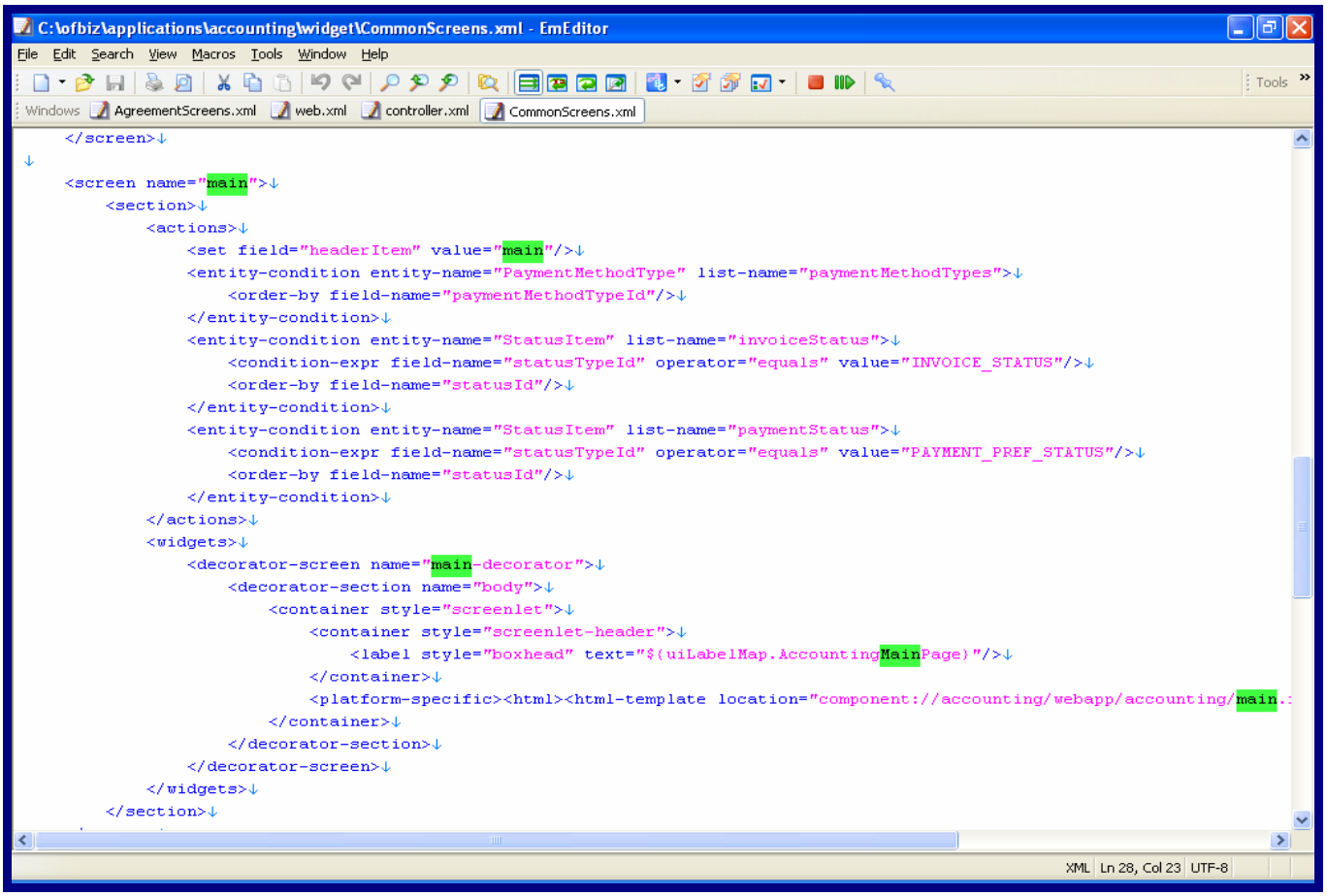


**Figure 31**

This is just a simple start, the structure if the screens will be easier to see in the coming examples.

*Back to the application :*

Now, we would visit the Agreements page. If you move the mouse above the “Agreemnts” tab you would notice that it makes a request to the “FindAgreement” map of the controller, as shown below.

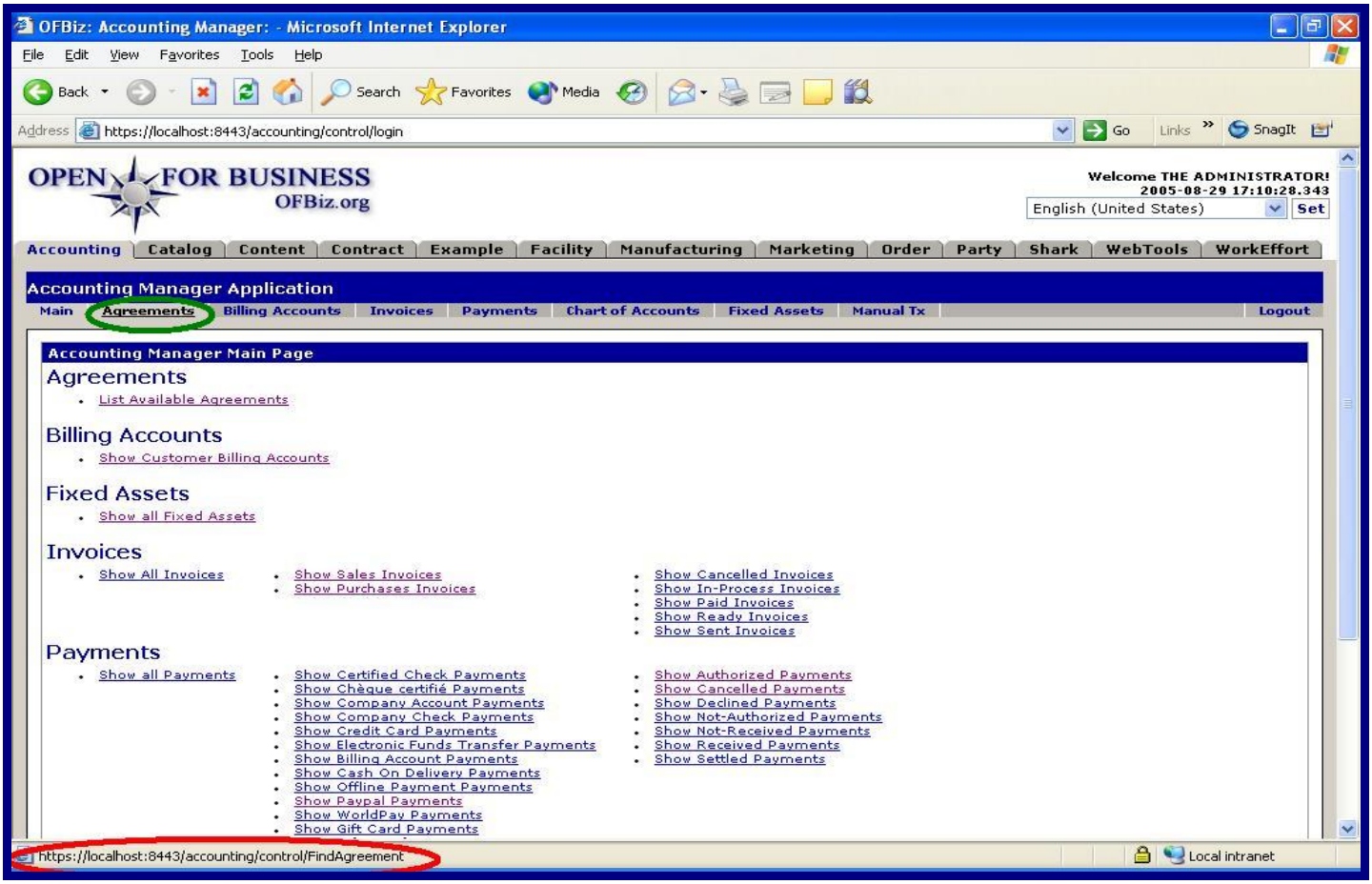


**Figure 32**

One click, and you will be forwarded to the FindAgreement page …

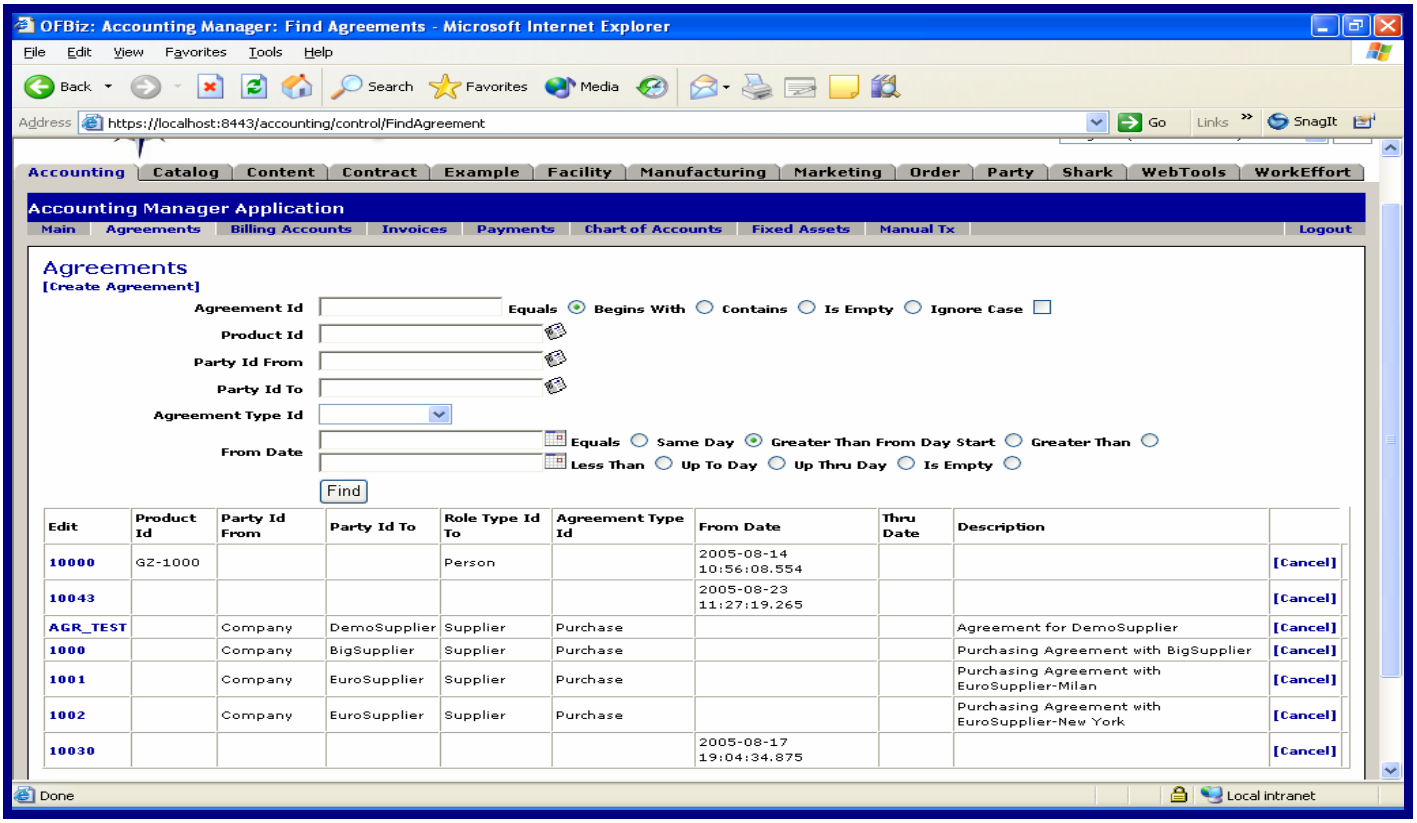


**Figure 33**

How that forwarding happened? We would follow it step by step :

1)Search the controller for the FindAgreement , and again the controller.xml is here :

C:\ofbiz\applications\accounting\webapp\accounting\WEB-INF or you can see it in the path in the Figure.

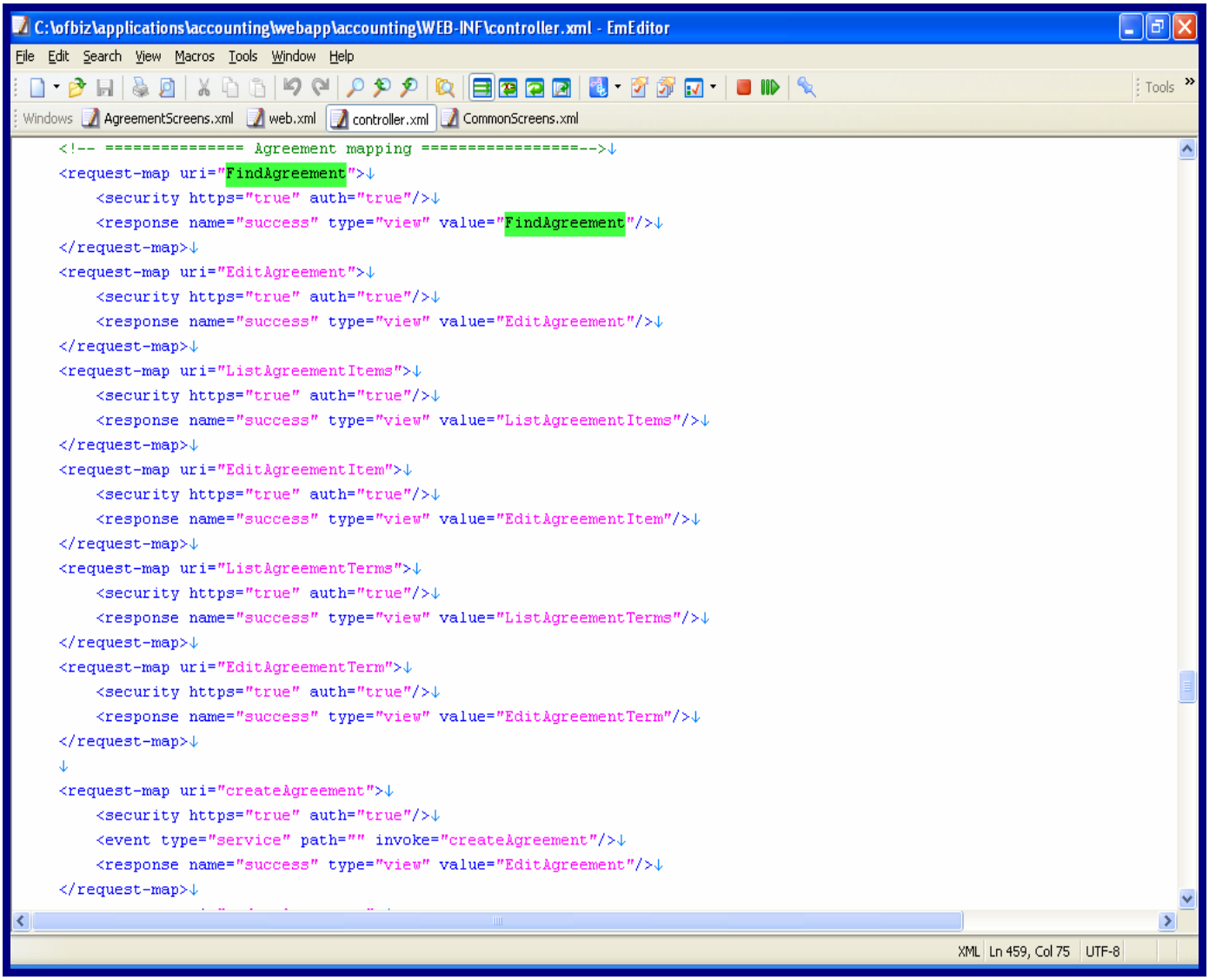


**Figure 34**

you will find that FindAgreement is of type view, and when the request succeded , you will be forwarded to the view whose value is : FindAgreement.

2) Still inside the controller, go down at the end of the file, and look for the view-map whose name is “FindAgreement” .

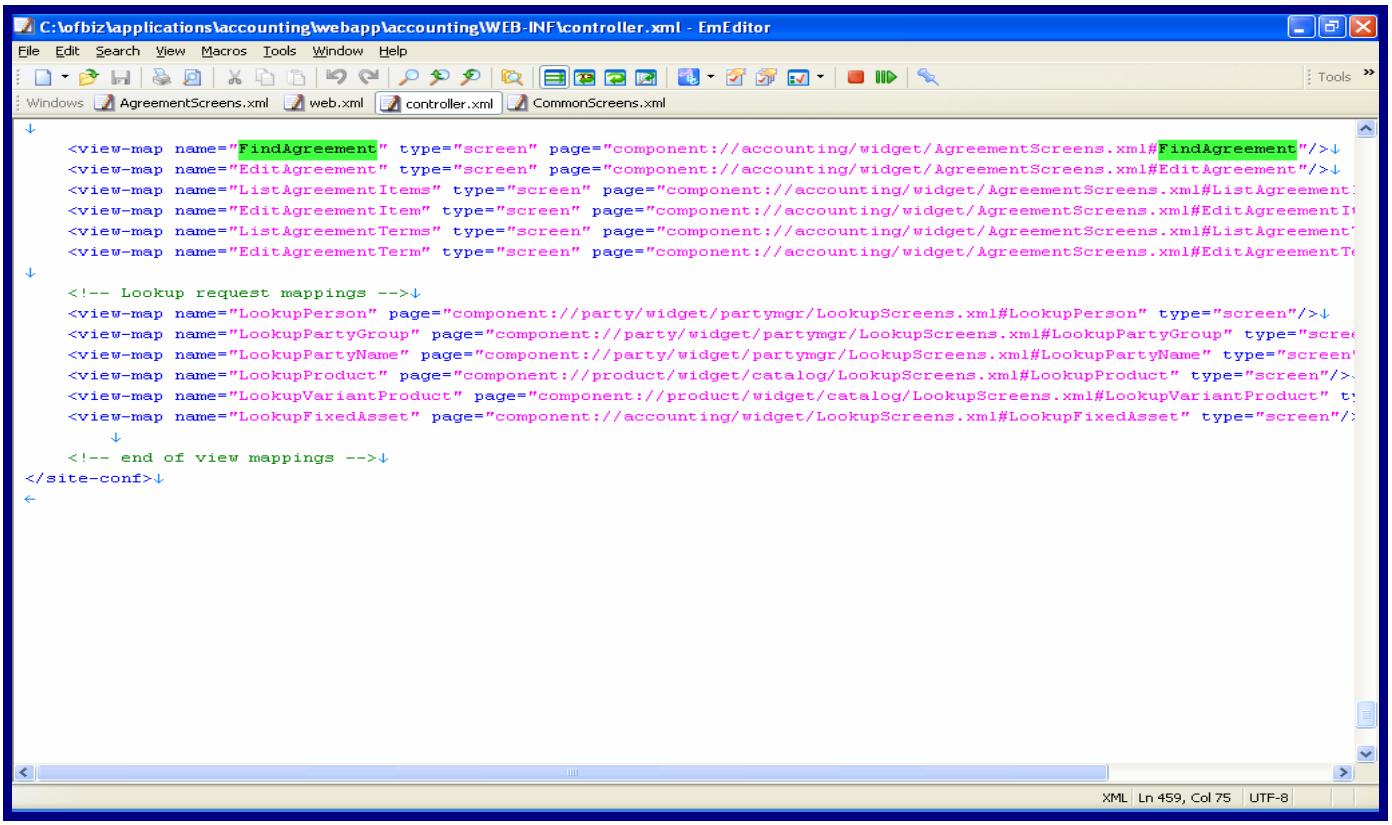


**Figure 35**

Now we would see that the FindAgreement view and is located in the AgreementScreens.xml file, in the #FindAgreement screen.

*Note :*

*The screen files, for example AgreementScreens.xml file, contain many screens inside it. So to determine which screen is the one to be rendered , it is stated in the controller after the symbol ‘#’ . Thus, in our case we know it is the screen FindAgreement inside the AgreementScreens.xml*

3) We would follow the path of the screen , provided in the controller, again the path appears in the header of each Figure.

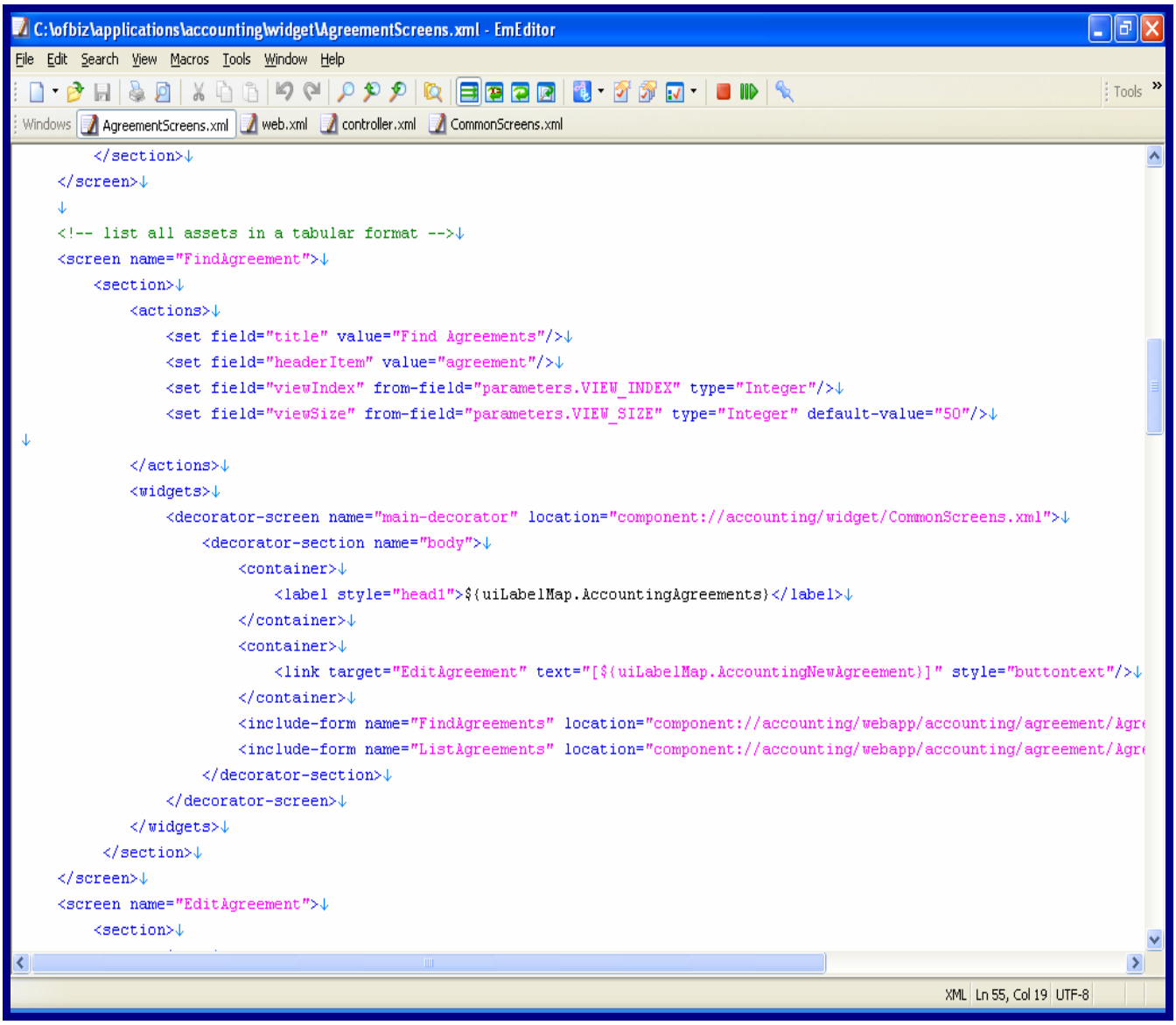


**Figure 36**

Notice that the FindAgreement screen uses the “main-decorator”, so all together form the page. Thus we will take each part in the page. To know more about the main-decorator, you could refer to the HelloWorld tutorials in here .

Now here is the FindAgreement page :

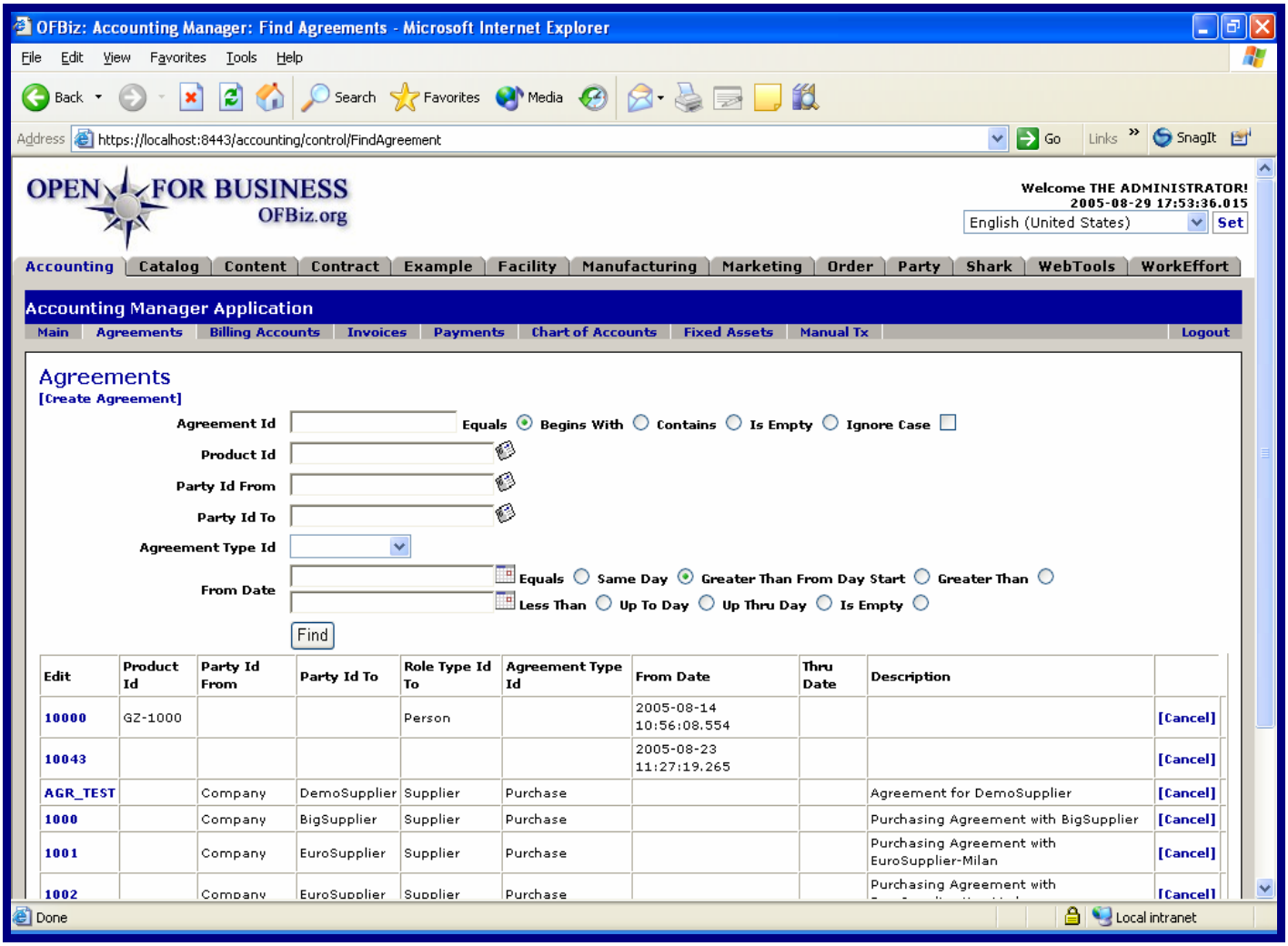


**Figure 37**

## The page is divided into many parts :

This is the application bar, and it contains all the OFBiz applications, the first one is the accounting application.


**Figure 38**

**This is the name of the Manager Application. This comes from the appheader.ftl file**


**Figure 39**

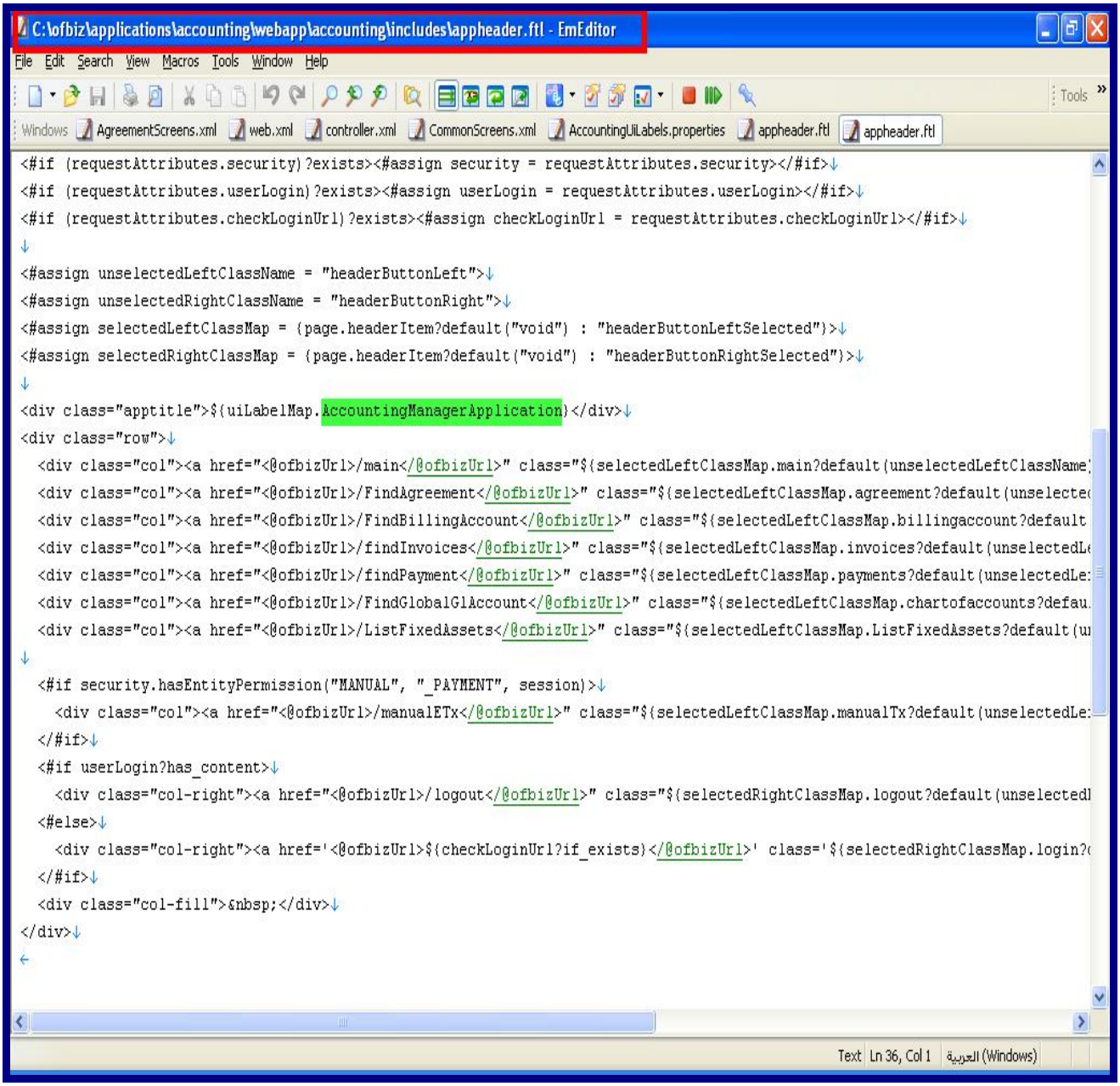


**Figure 40**

Note that it uses the uiLabelMap.AccountingManagerApplication, i.e., it is reading the name “Accounting Manager Application” from the uiLabelMap. This is located in the config directory : C:\ofbiz\applications\accounting\config

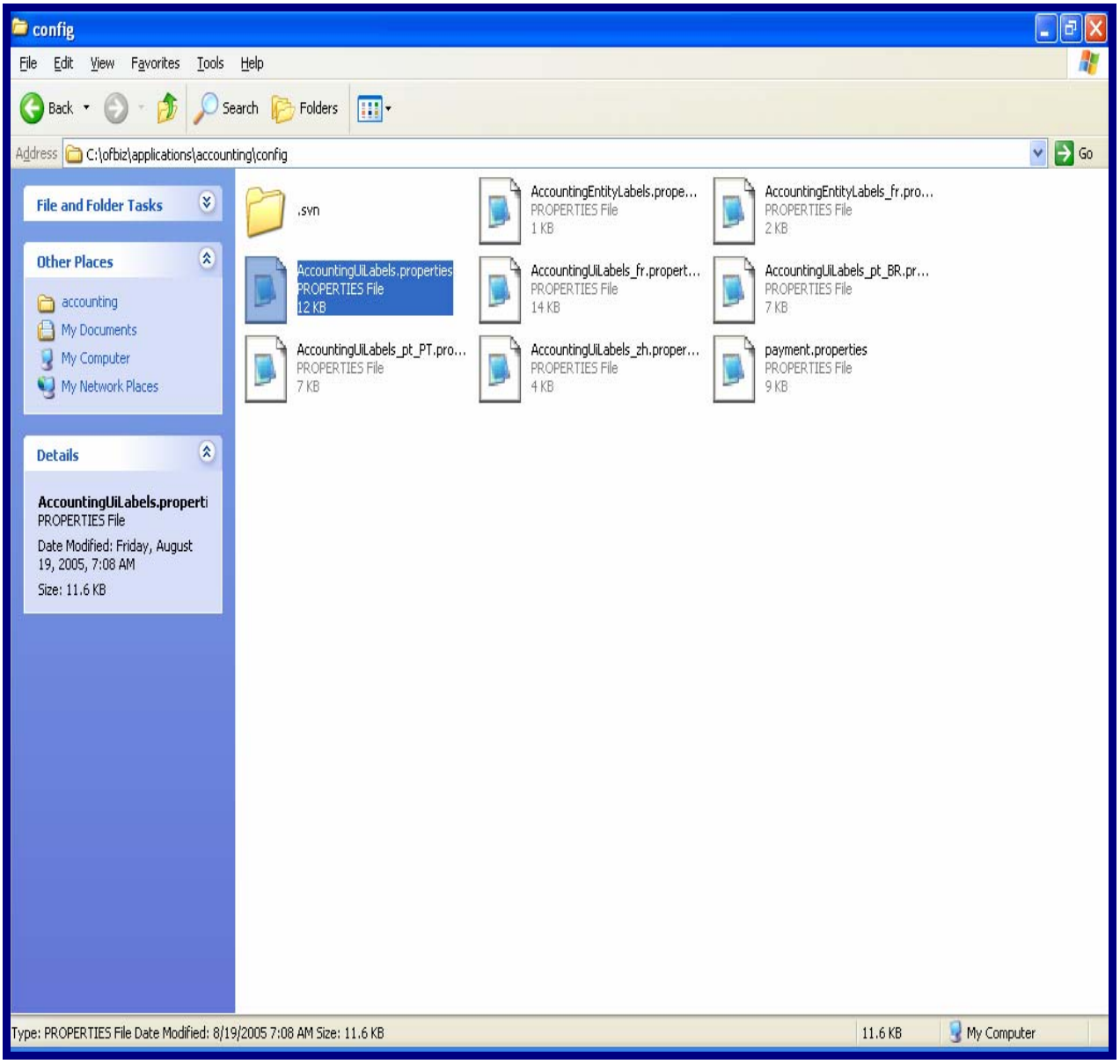


**Figure 41**

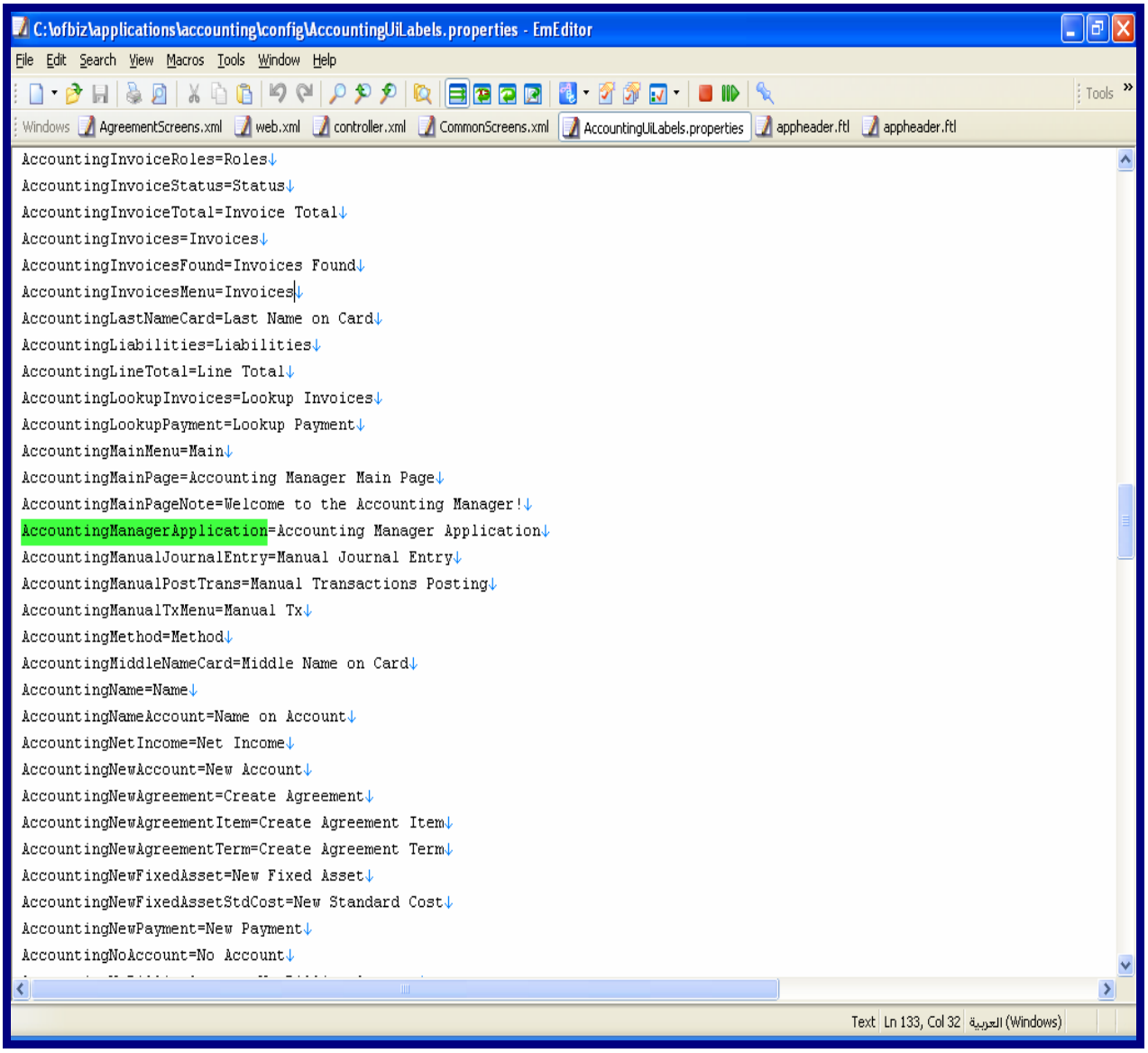


**Figure 42**

Note that this is needed because different languages would use different user interface, so this would also be found in French, German, ..etc .

---

This is also from the appheader.ftl file.

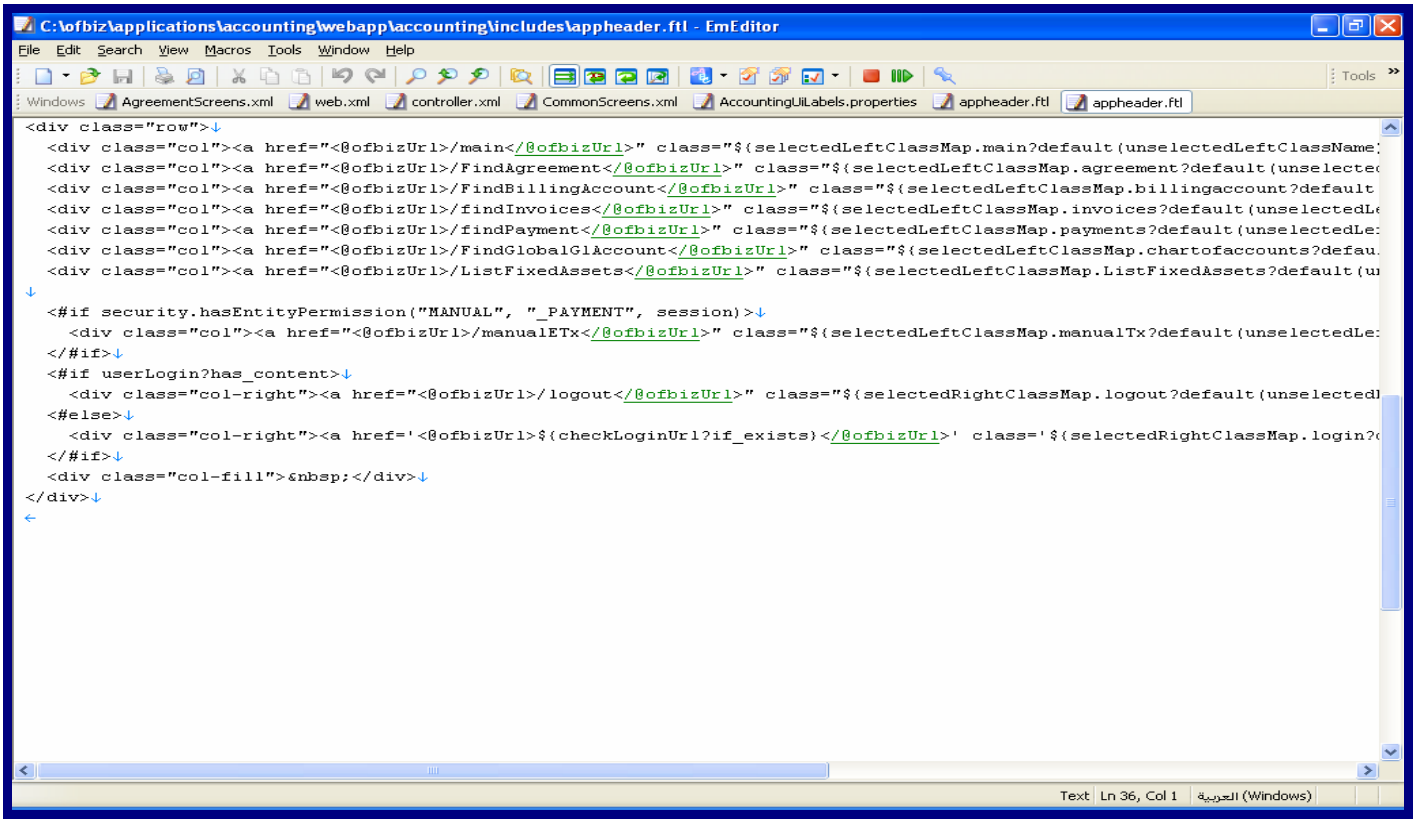


**Figure 43**

---

After that , the rest of the page is coming from the <widget> section of the FindAgreement screen.

```
<widgets>
    <decorator-screen name="main-decorator" location="component://accounting/widget/CommonScreens.xml">
        <decorator-section name="body">
            <container>
                <label style="head1">${uiLabelMap.AccountingAgreements}</label>
            </container>
            <container>
                <link target="EditAgreement" text="[${uiLabelMap.AccountingNewAgreement}]" style="buttontext"/>
            </container>
            <include-form name="FindAgreements" location="component://accounting/webapp/accounting/agreement/Agr
            <include-form name="ListAgreements" location="component://accounting/webapp/accounting/agreement/Agr
        </decorator-section>
    </decorator-screen>
</widgets>
```

**Figure 44**

---

Agreements
[Create Agreement]

This text “Agreements” is coming from the <container> tag, the uiLabelMap.AccountignAgreements provide us with the word “Agreement” with different languages.

The [Create Agreement] comes from the container that has a link to the EditAgreement page.

---

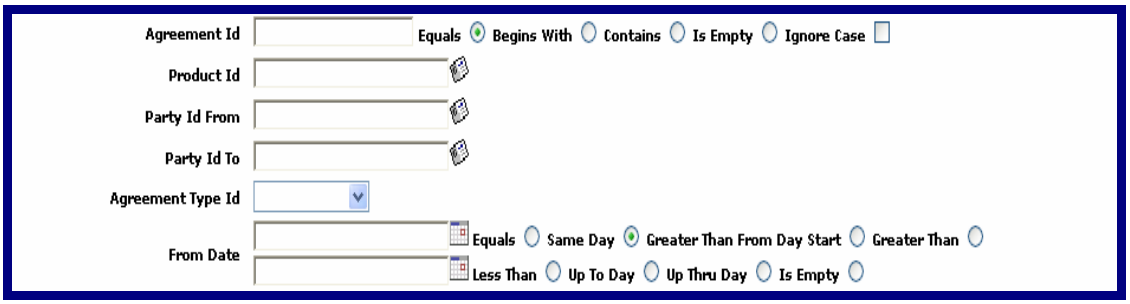


**Figure 45**

This part comes from the included form, FindAgreements :

```
<include-form name="FindAgreements" location="component://accounting/webapp/accounting/agreement/AgreementForms
```

**Figure 46**

---

while :

| Edit | Product Id | Party Id From | Party Id To | Role Type Id To | Agreement Type Id | From Date | Thru Date | Description | |
|---|---|---|---|---|---|---|---|---|---|
| 10000 | GZ-1000 | | | Person | | 2005-08-14 10:56:08.554 | | | [Cance |
| 10043 | | | | | | 2005-08-23 11:27:19.265 | | | [Cance |
| AGR_TEST | | Company | DemoSupplier | Supplier | Purchase | | | Agreement for DemoSupplier | [Cance |
| 1000 | | Company | BigSupplier | Supplier | Purchase | | | Purchasing Agreement with BigSupplier | [Cance |
| 1001 | | Company | EuroSupplier | Supplier | Purchase | | | Purchasing Agreement with EuroSupplier-Milan | [Cance |
| 1002 | | Company | EuroSupplier | Supplier | Purchase | | | Purchasing Agreement with EuroSupplier-New York | [Cance |
| 10030 | | | | | | 2005-08-17 19:04:34.875 | | | [Cance |

This part comes from the included form : ListAgreements:

```
<include-form name="ListAgreements" location="component://accounting/webapp/accounting/agreement/AgreementForms.xm
```

**Figure 47**

---

Now we would have a look at the Forms , that are located in the directory :

C:\ofbiz\applications\accounting\webapp\accounting\agreement
\AgreementForms.xml

Here is the FindAgreements form :

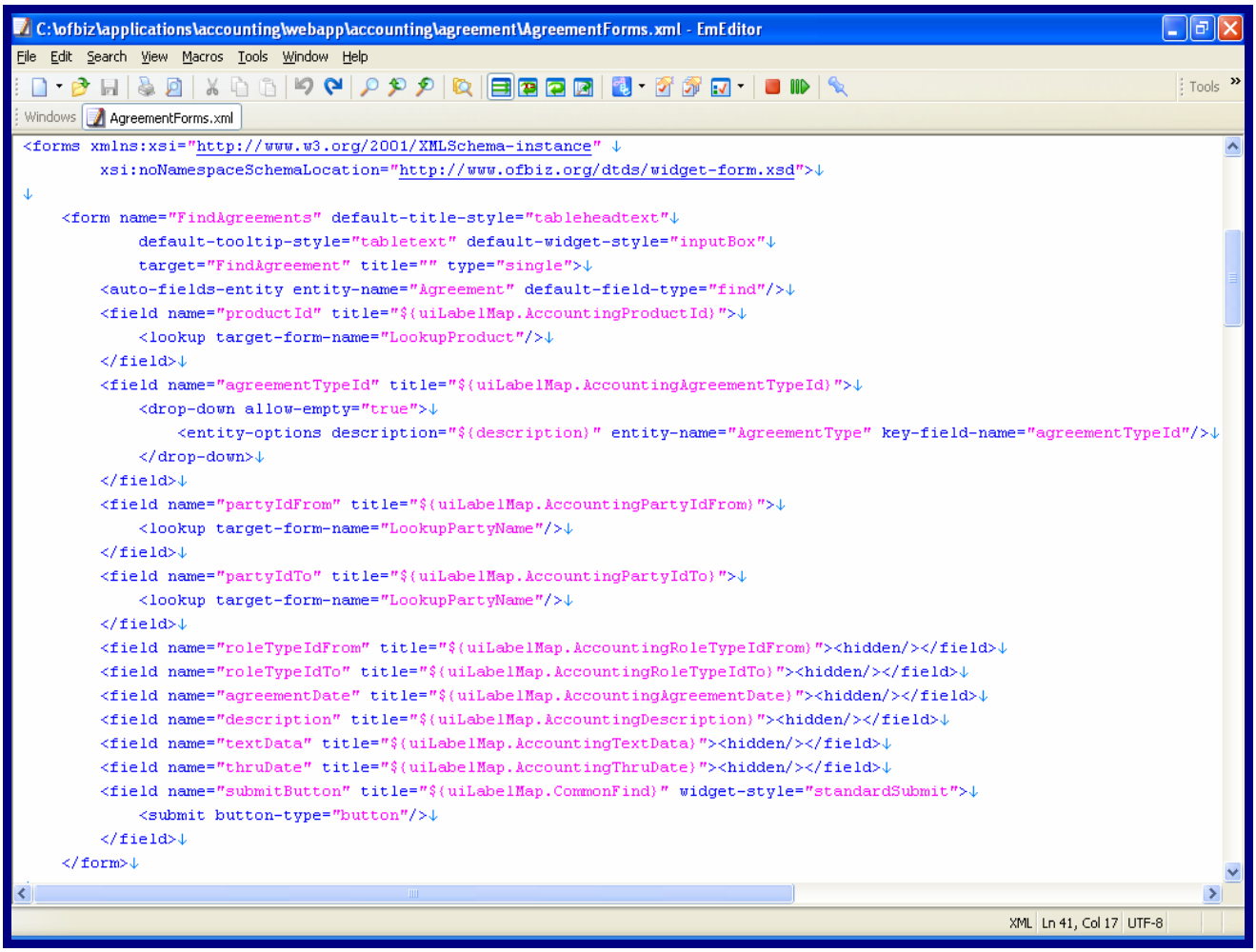


**Figure 48**

Notice that a lot of tags are used, for the time being, you need just to know the most important ones that serve your application.

```
target="FindAgreement" title="" type="single">
```

**Figure 49**

Shows to which page is the next target after serving the form.

```
<auto-fields-entity entity-name="Agreement" default-field-type="find"/>
```

**Figure 50**

This tag will by default,read all the fields of the Agreement entity, and display them in the same format based on the default-field-type , unless the field is explicitly specified to have a different feature as we would see.
Note that the “find” field type is the default, and it is among four different field types , find , edit, display and hidden, try changing the “find” to any of those and see what changes would occur, don’t worry it is an open source project!

Here is the Agreement Entity and its fields :

```
    <entity entity-name="Agreement"
            package-name="org.ofbiz.party.agreement"
            title="Agreement Entity">
      <field name="agreementId" type="id-ne"></field>
      <field name="productId" type="id"></field>
      <field name="partyIdFrom" type="id"></field>
      <field name="partyIdTo" type="id"></field>
      <field name="roleTypeIdFrom" type="id"></field>
      <field name="roleTypeIdTo" type="id"></field>
      <field name="agreementTypeId" type="id"></field>
      <field name="agreementDate" type="date-time"></field>
      <field name="fromDate" type="date-time"></field>
      <field name="thruDate" type="date-time"></field>
      <field name="description" type="description"></field>
      <field name="textData" type="long-varchar"></field>
```

**Figure 51**

Unless a field is explicitly specified, all the field will appear with a “find” filed type.

---

```
<field name="productId" title="${uiLabelMap.AccountingProductId}">
    <lookup target-form-name="LookupProduct"/>
</field>
```

**Figure 52**

Here is the productId field , it is explicelty specified to be labeled with the label AccountingProductId . You could also just write the name like title=”Product Id” .However they would use to read from the config files, to support different lganuage as mentioned earlier.
It also uses another defined tag called lookup , to allows to see the product information to choose among them.

```
<field name="agreementTypeId" title="${uiLabelMap.AccountingAgreementTypeId)">↓
    <drop-down allow-empty="true">↓
        <entity-options description="${description)" entity-name="AgreementType" key-field-name="agreementTypeId"/>↓
    </drop-down>↓
</field>↓
```

**Figure 53**

The agreementTypeId field of the agreement entity is explicitly defined to have a label , using the title=”…”
Also, it is to be shows as a drop-down menu .
The <drop-down> has an attribute called allow-empty = “true” which means it can be empty, else if “false” then it has to have one of the types.

<entity-options> tells that the filed “agreementTypeId” will have the values from the AgreementType entity , and it will match it with the field “agreementTypeId of the AgrementType entity.
What will be shown to the users is the “description” field of the AgreementType entity that describes a particular agreementTypeId.

---

```
<field name="roleTypeIdFrom" title="${uiLabelMap.AccountingRoleTypeIdFrom)"><hidden/></field>↓
<field name="roleTypeIdTo" title="${uiLabelMap.AccountingRoleTypeIdTo)"><hidden/></field>↓
<field name="agreementDate" title="${uiLabelMap.AccountingAgreementDate)"><hidden/></field>↓
<field name="description" title="${uiLabelMap.AccountingDescription)"><hidden/></field>↓
<field name="textData" title="${uiLabelMap.AccountingTextData)"><hidden/></field>↓
<field name="thruDate" title="${uiLabelMap.AccountingThruDate)"><hidden/></field>↓
```

**Figure 54**

This list of fields are specified to be hidden, so clearly they will not appear in the page with other fileds.

---

Finally,

```
<field name="submitButton" title="${uiLabelMap.CommonFind)" widget-style="standardSubmit">↓
    <submit button-type="button"/>↓
</field>↓
```

**Figure 55**

The submit button to submit the form .

If you followed the ListAgreement form, you would notice it is similar to the findAgreement , so I won’t discuss it here .I would discuss now the EditAgreement page/form.

---

## Edit Agreement :

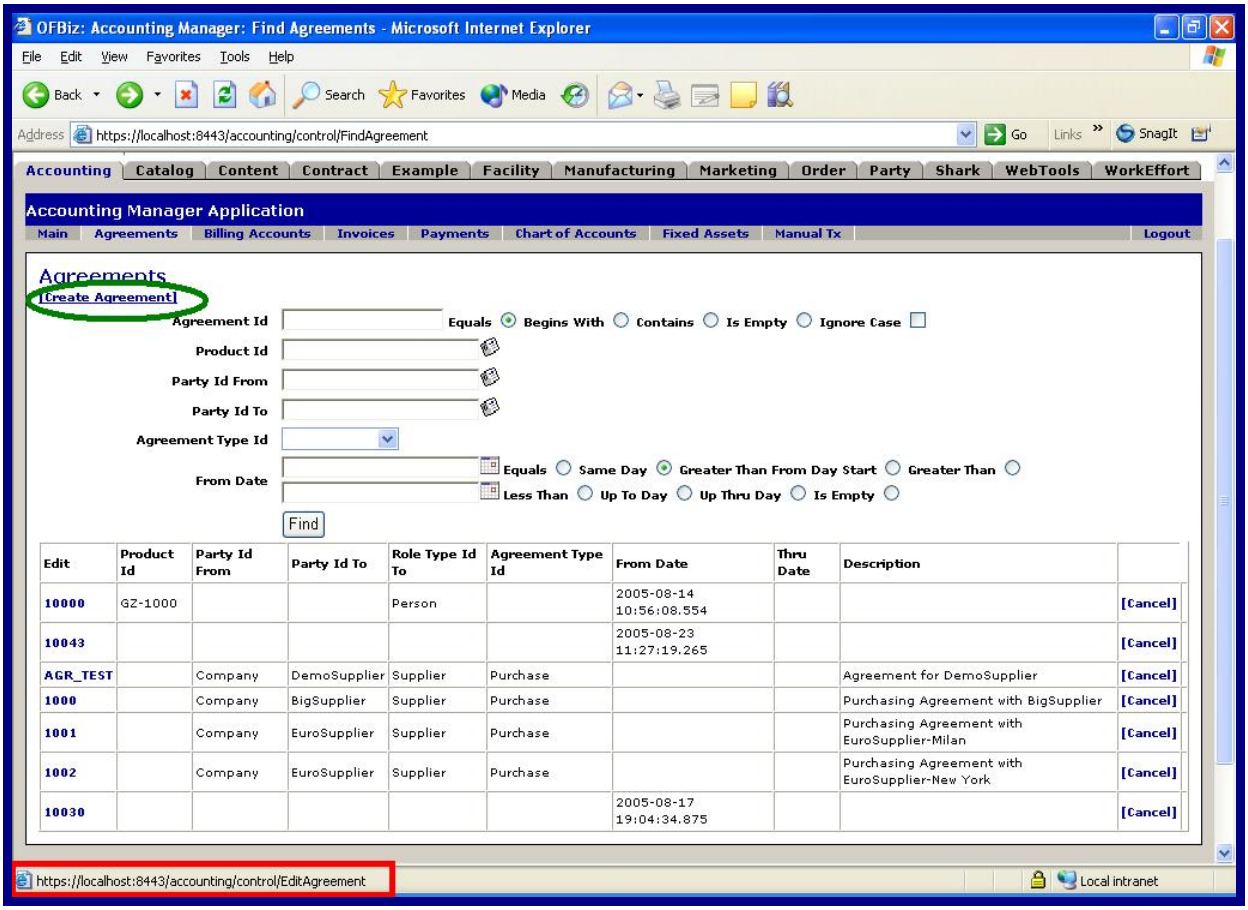


**Figure 56**

Again, and exactly as we did with the FindAgreement ,if we want to press on the [create Agreement] , we would notice a request to control/EditAgreement, and again, and if we followed the controller it will guide us to the EditAgreement screen in the AgreementScreens.xml file.

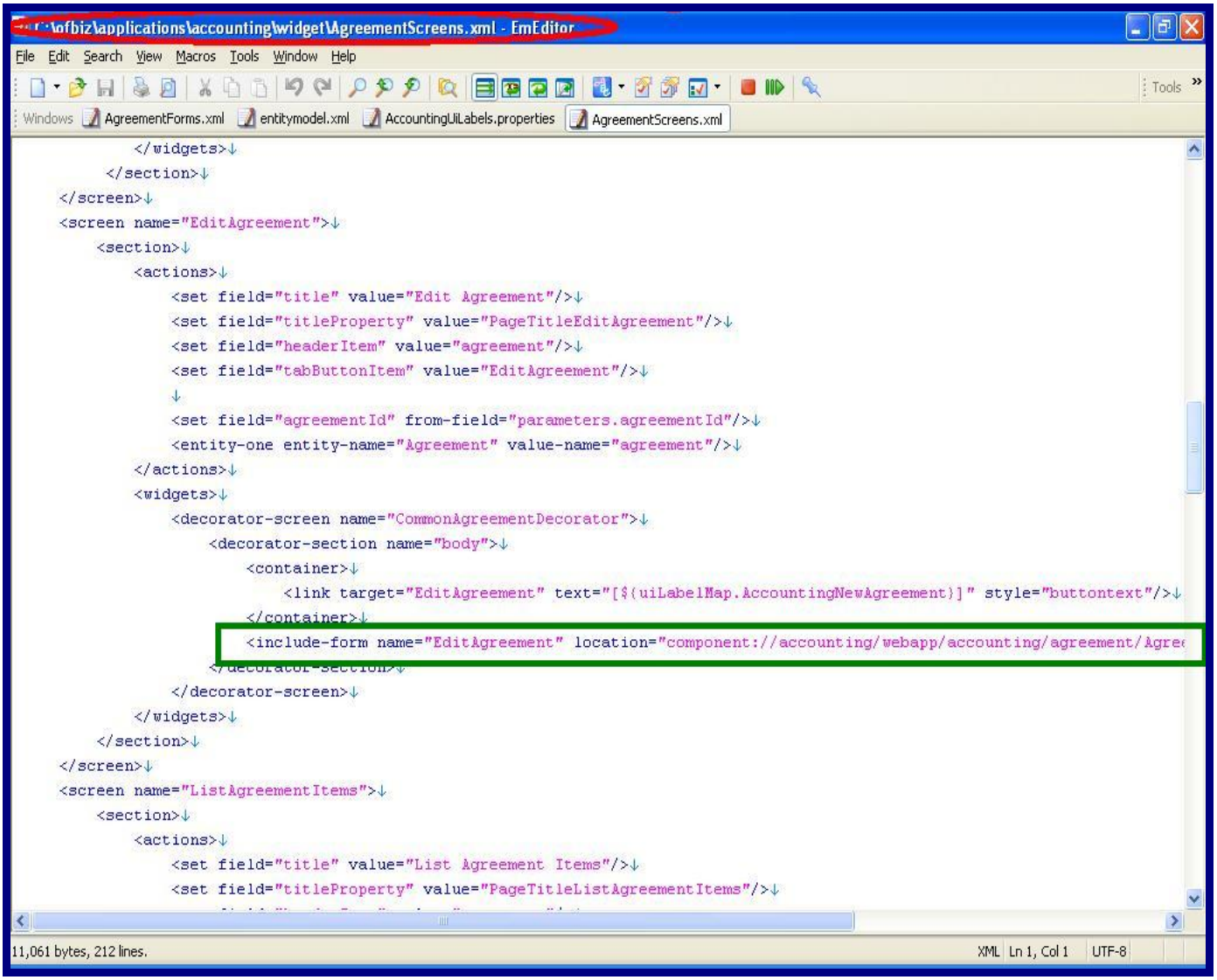


**Figure 57**

Figure 57 shows the Edit Agreement screen , again similar to the FindAgreement screen.

It is also including the EditAgreement form, so whenever a request to this page is rendered, the form will be included in the page.

Here is the EditAgreement form :

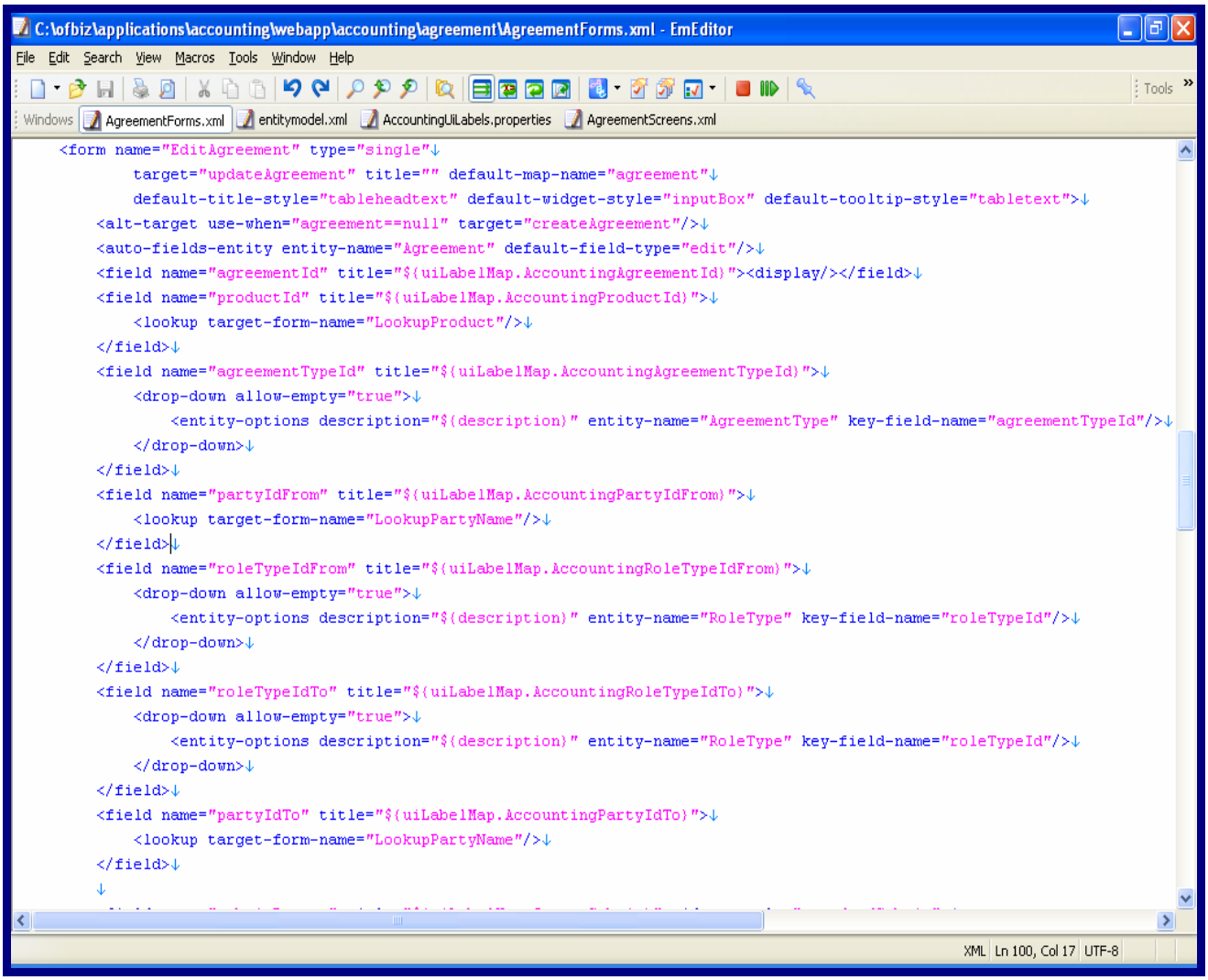


**Figure 58**

```
target="updateAgreement" title="" default-map-name="agreement"↓
```

**Figure 59**

It says that its next target should be updateAgreement, but what is updateAgreement? Where is it ? the Controller would “always” answer .

But it also says :

```
<alt-target use-when="agreement==null" target="createAgreement"/>↓
```

**Figure 60**

It is clear now that if the agreement we are dealing with is “new” ,i.e., =null , so our target will be “createAgreement” , else our target is “updateAgreement” .

The rest of this form is a normal declaration for the fileds , as we saw earlier in the FindAgreement form.

Let us now follow the createAgreement and the updateAgreement.

1)Assume we are creating a new agreement, that is we pressed on the button [create Agreement] in Figure 56 , and started entering the fields of the EditAgrement .
Thus, it will it find that the condition “agremment == null” returns true .

So, the request goes to the controller , for createAgreement.

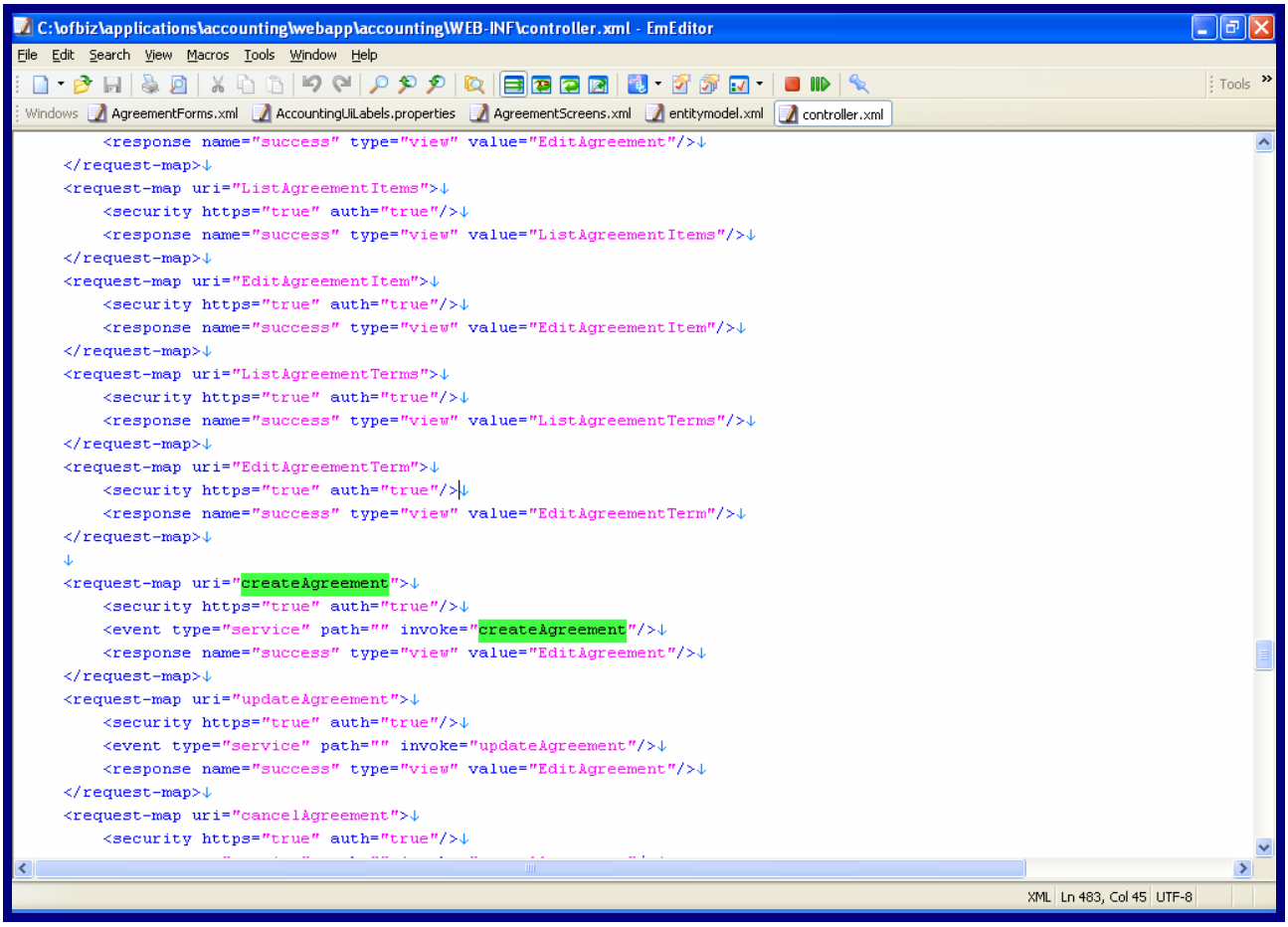


**Figure 61**

We found the createAgreement , but it is of type service! “Not a view”.

and it invokes a service called createAgreement , and on success , i.e., if we didn't have any errors or problems, it would be redirected to the EditAgreement page.

```
<event type="service" path="" invoke="createAgreement"/>
```

**Figure 62**

---

Now , and by default, the controller will try to search for the createAgreement service , in the servicedef directory. It will then find the definition for this service" createAgreement" in the service_agreement.xml file .

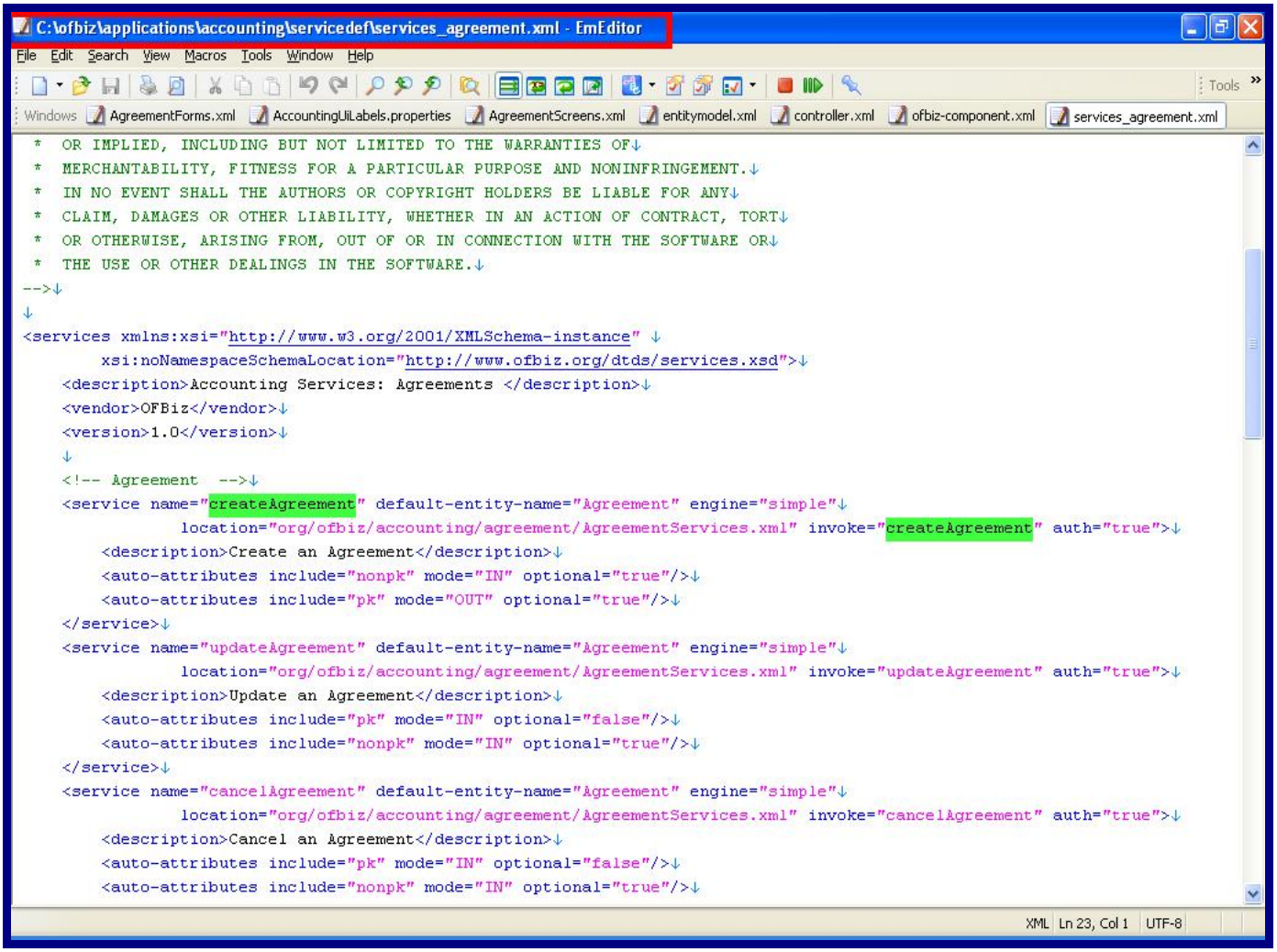


**Figure 63**

Here it is clear, the createAgreement service will be requested It will be servicing the Agreement entity as mentioned in the default-entity-name .
It is of type simple, it will invoke a simple method called createAgreement, and its location is provided.

The <auto-attributes> will govern the mode and existence of the non-pk and pk attributes of the entity Agreement.

You can follow the again refer to the service engine document on the OFBiz website.

Then we would follow the path , to see the implementation of the createAgreement simple method.

And finally here is the createAgreement service implemented with the OFBiz mini-language.

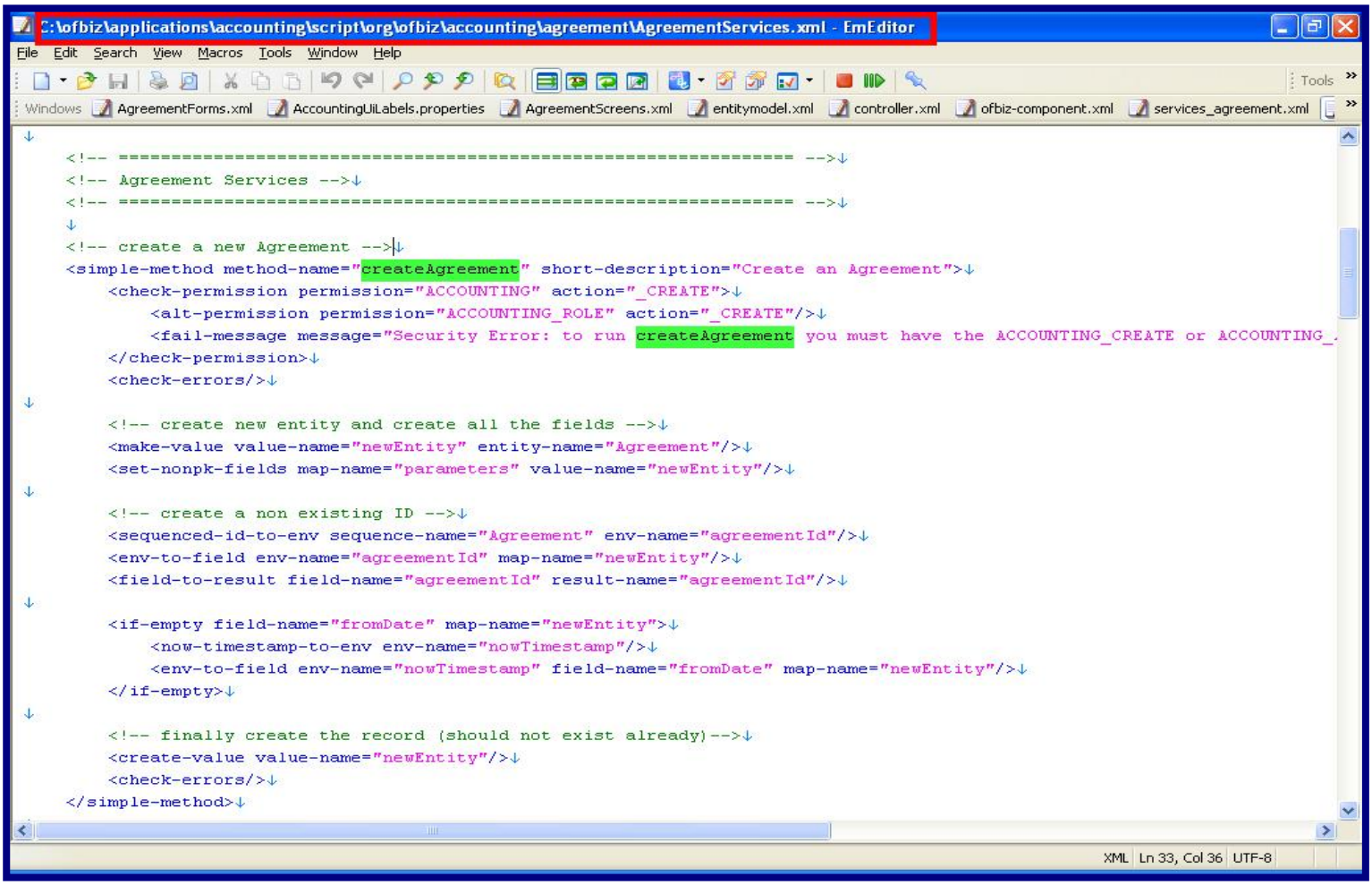


**Figure 64**

```
<!-- create new entity and create all the fields -->
<make-value value-name="newEntity" entity-name="Agreement"/>
<set-nonpk-fields map-name="parameters" value-name="newEntity"/>
```

**Figure 65**

As shown,
Make-value means we want to create a new row in the databaseof type "Agreement". Parameters "fields" are sent/sent/get using Maps. So, now our map that will contain the fields of the new created entity is called . "newEntity" .

This "newEntity" map is taking the values of all the non-pk from another map called "parameters" .The parameters map holds the values/parameters of the form, "i.e., from the parameters entered by the user in the form ".

---

```
<!-- create a non existing ID -->
<sequenced-id-to-env sequence-name="Agreement" env-name="agreementId"/>
<env-to-field env-name="agreementId" map-name="newEntity"/>
<field-to-result field-name="agreementId" result-name="agreementId"/>
```

**Figure 66**

Here is is creating the sequence for the agreementId field which is the primary key of the Agreement entity, then it is filling the agreementId in the "newEntity" map.

---

```
<if-empty field-name="fromDate" map-name="newEntity">
    <now-timestamp-to-env env-name="nowTimestamp"/>
    <env-to-field env-name="nowTimestamp" field-name="fromDate" map-name="newEntity"/>
</if-empty>
```

**Figure 67**

The filed fromDate should not be null "you do not expect an agreement not to have the date taken" . Thus, if the user didn't fill it, it will be filled with the current time value, using the "nowTimesamp" env-name .

---

```
    <!-- finally create the record (should not exist already)-->↓
    <create-value value-name="newEntity"/>↓
    <check-errors/>↓
</simple-method>↓
```

**Figure 68**

And finally, the new row is created in the database.

---

Here is the createAgreement :

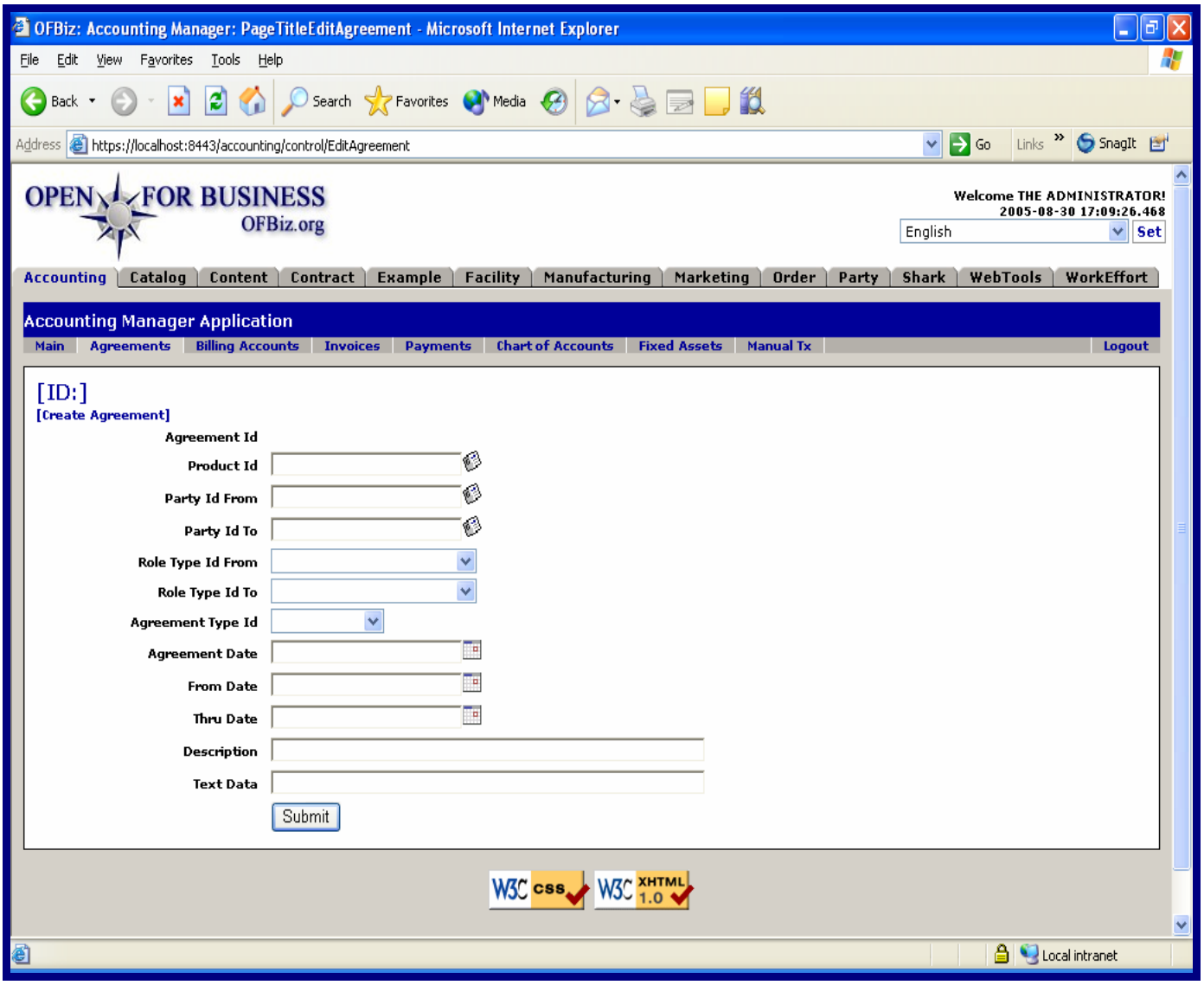


**Figure 69**

And here after submitting the button “note that we saw that button in at the end of the “EditAgreement” form.

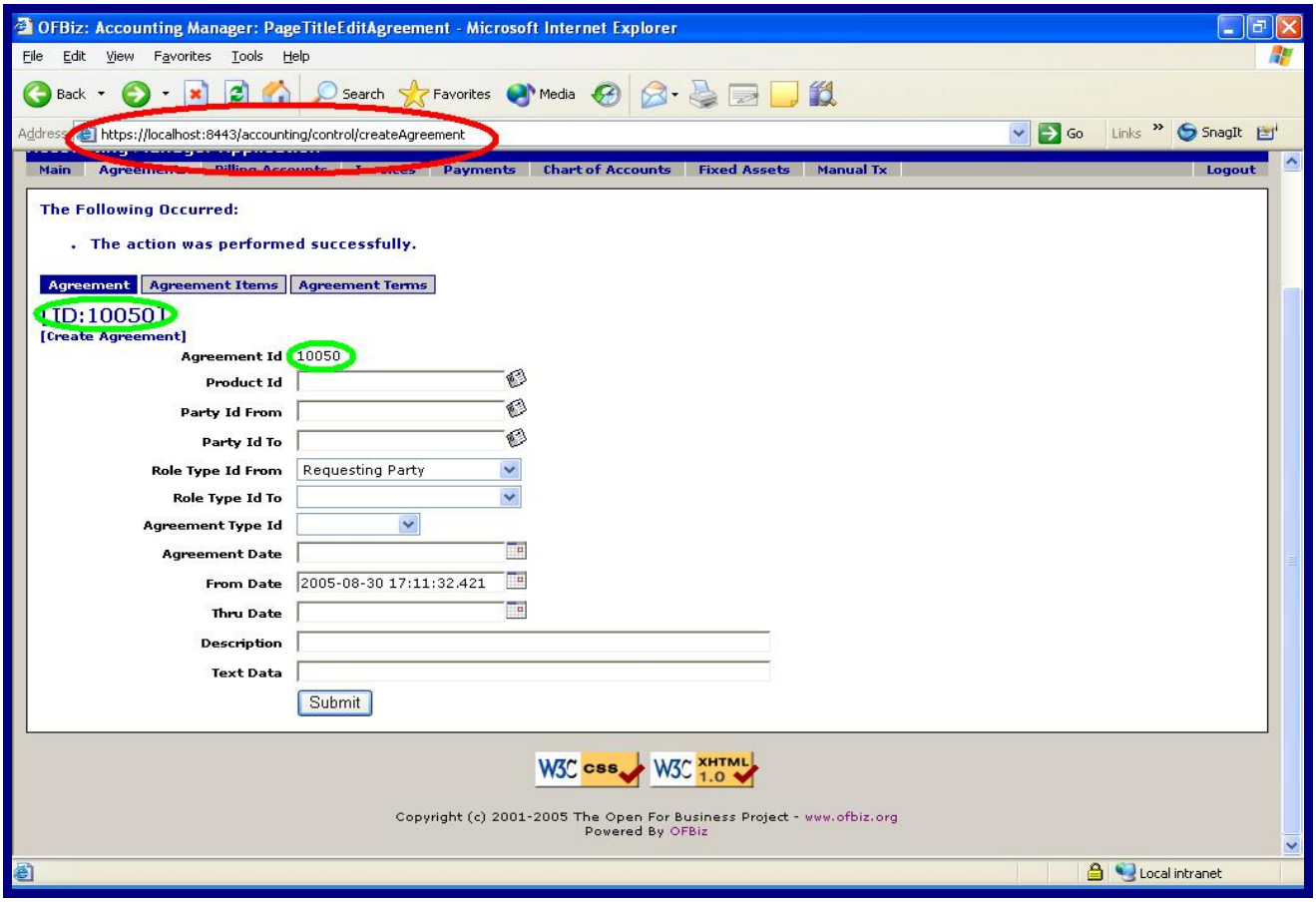


**Figure 70**

Note the control/createAgreement request , because the agreement== null was true, since the user hasn’t created the agreement yet.

Now, and after we have created a new row in the database , in the table Agreement , whose ID is 10050 . Now a click on the submit button, will check and find that agreement == null is not a true anymore, since the agreement will read the values of the fields from the “parameters” map . Thus it will request the control/updateAgreement .Again the cycle will be reapted :
controller > updateAgreement is a service> looks for it in the servicedef > finds out its type and location > looks for its implementation > perform the service >data stored in the database > gets back to the page as shown below.

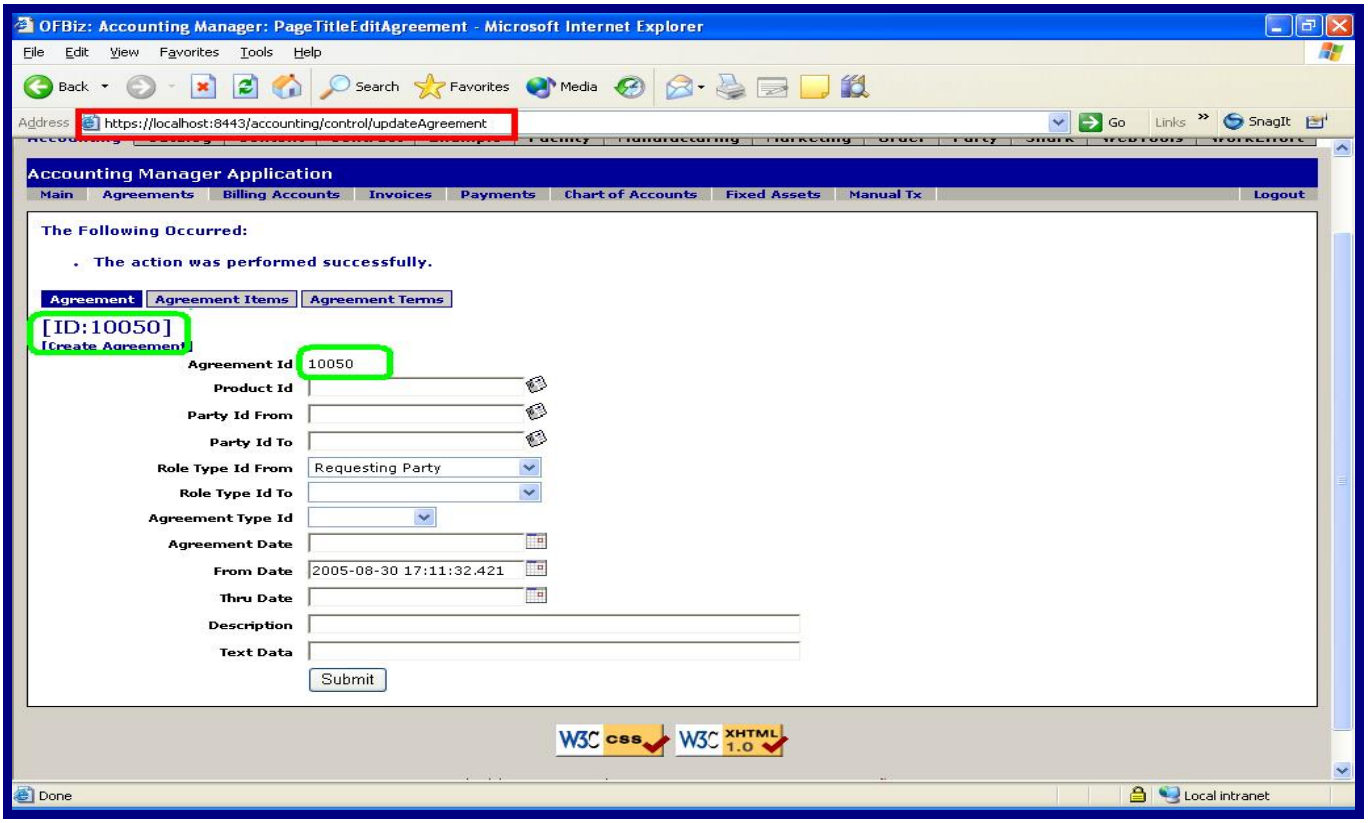


**Figure 71**

Notice that : these


**Figure 72**

were not there in the createAgreement Figure,Figure 69, so they must be attached to a particular Agreement , in other words, there should be an agreement , for them to appear.

So we would chech the AgreementScreen again, we would find it is using main-decorator pattern,

```
<screen name="EditAgreement">
    <section>
        <actions>
            <set field="title" value="Edit Agreement"/>
            <set field="titleProperty" value="PageTitleEditAgreement"/>
            <set field="headerItem" value="agreement"/>
            <set field="tabButtonItem" value="EditAgreement"/>

            <set field="agreementId" from-field="parameters.agreementId"/>
            <entity-one entity-name="Agreement" value-name="agreement"/>
        </actions>
        <widgets>
            <decorator-screen name="CommonAgreementDecorator">
                <decorator-section name="body">
                    <container>
                        <link target="EditAgreement" text="[${uiLabelMap.AccountingNewAgreement}]" style="buttonte
                    </container>
                    <include-form name="EditAgreement" location="component://accounting/webapp/accounting/agreemen
```

**Figure 73**

and having a look into the main-decorator pattern , we would see it is including an ftl file called AgreementTabBar.ftl .

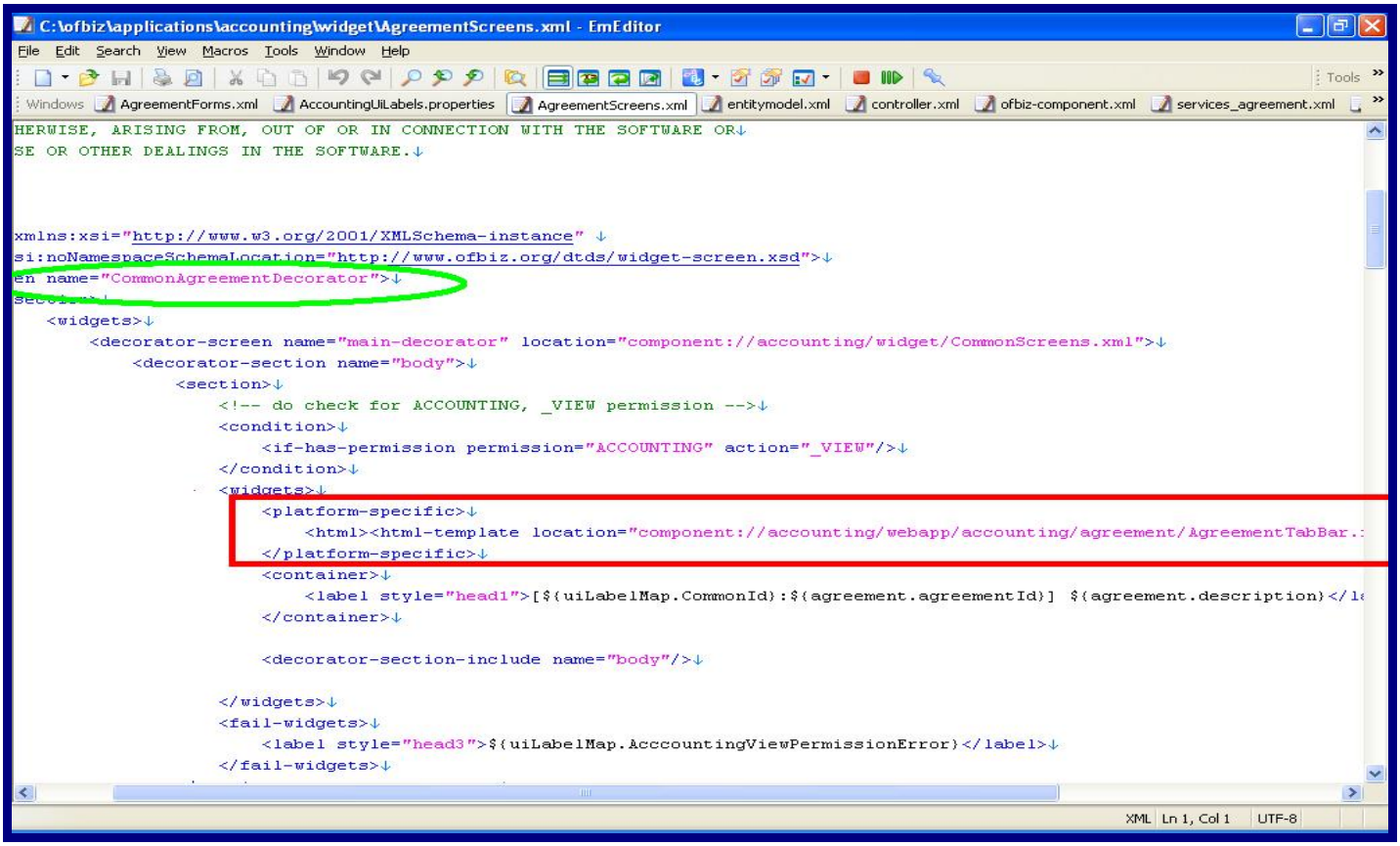


**Figure 74**

and inside the AgreementTabBar.ftl file :

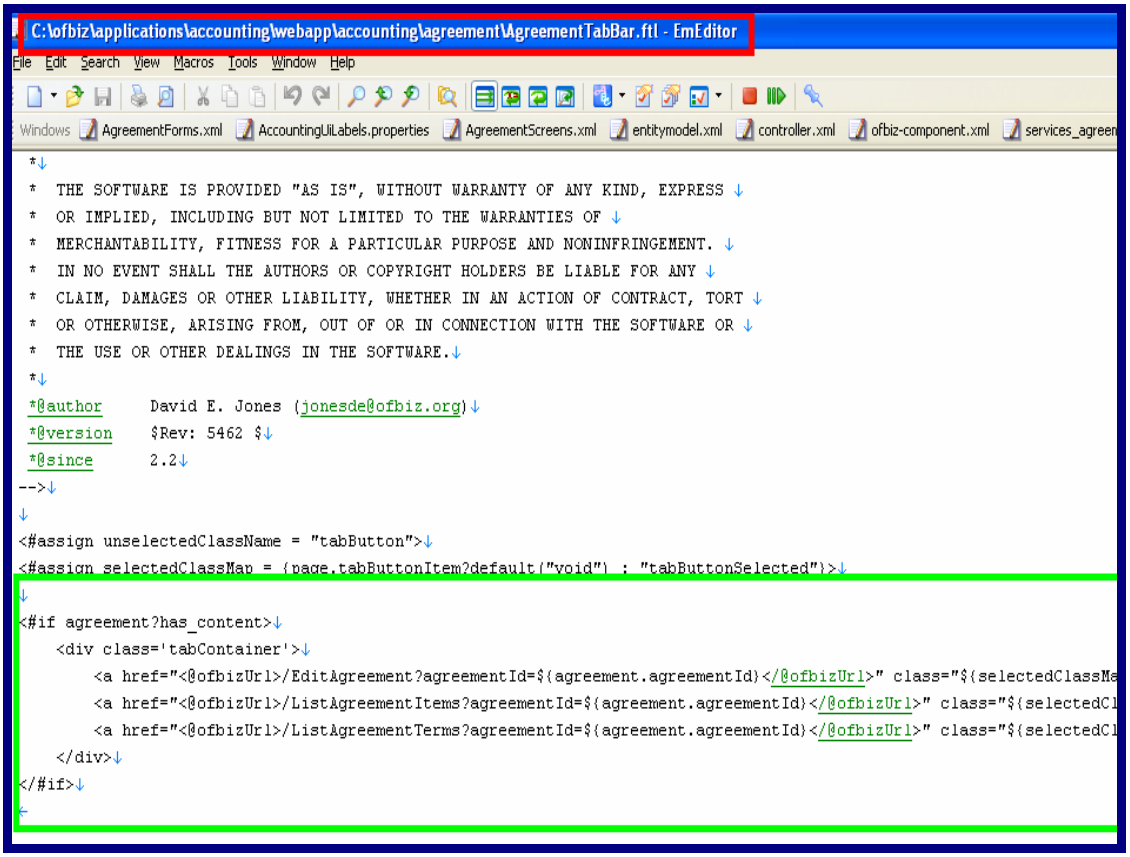


**Figure 75**

It is clear that these would appear only when there is an agreement because it is checking if the agremment has content ,i.e., it is not null, and it shows them" the button we have seen" only when this condition satisfies.
As for the ID : 10050 That had a green circle in Figure 72 , it also came from the commonAgreementDecorator :

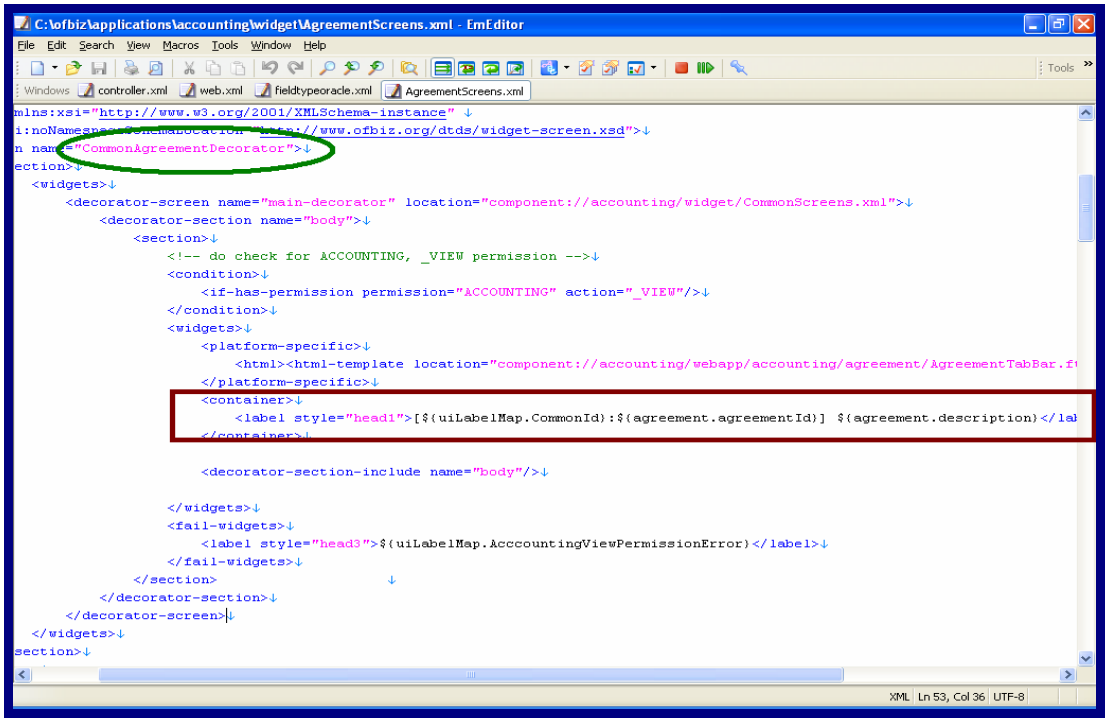


**Figure 76**

Finally , we would see this Agreement in our list.

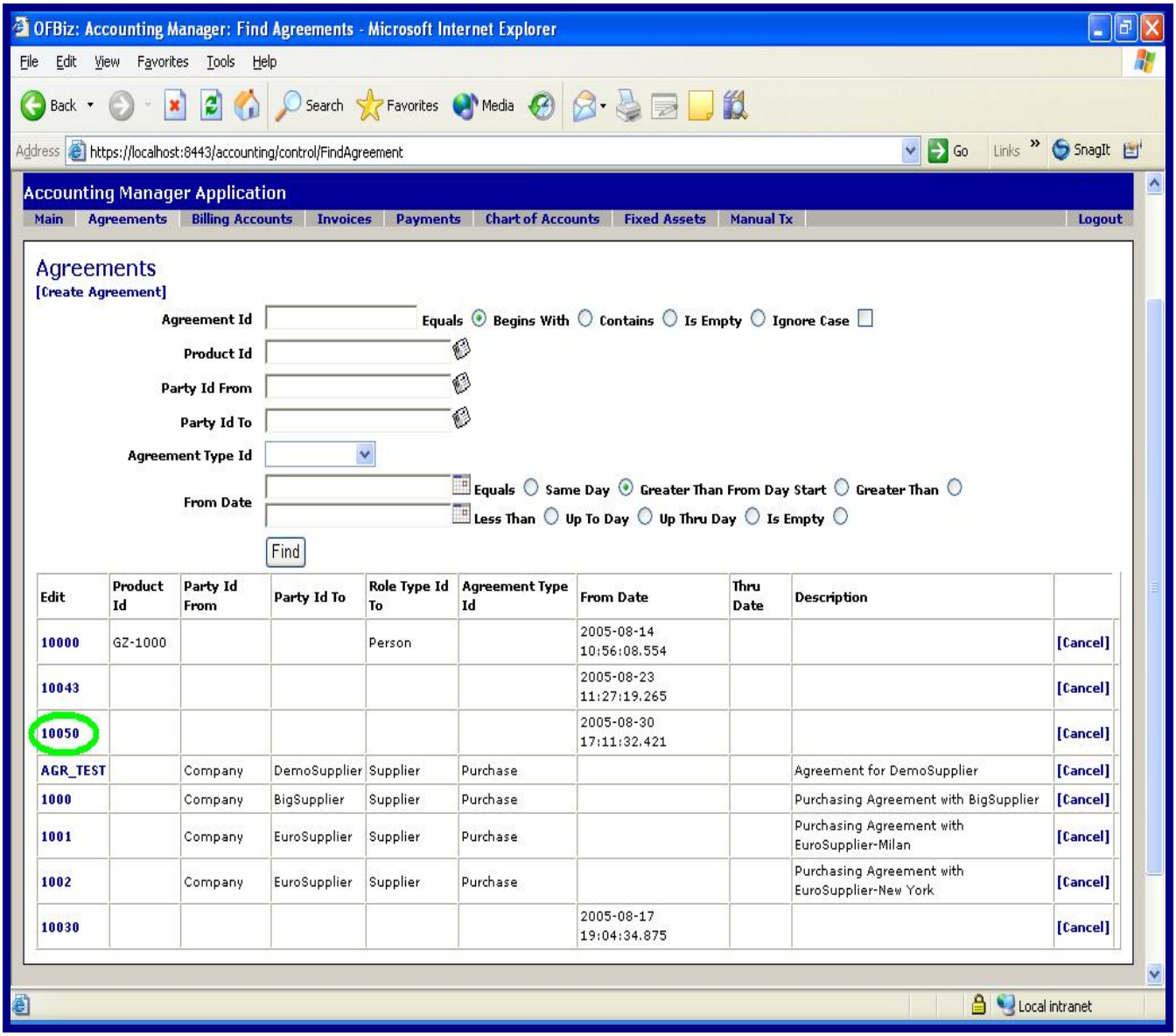


**Figure 77**

**Summary :**

OFBiz uses the three-tiers architecture in its model and a controller "main-servlet" to control and forward the requests to the application, whether this request is for a particular service or page. For building any application you need to build up these layers.

1) Data layer : represents the database, your stored data.

➔Design your database tables
➔Build the data layer in the entitydef folder:
➔ entitygroup.xml file to define the tables name.
➔ entitymodel.xml to implement the defined tables, their pk's, field types and relationships.

2) Business Logic Layer: represents what services applied on the database.

➔ Decide what services you need.
➔ Build this layer:
➔ servicedef folder : define all the services ,their types, the methods they invoke, their inputs and ouputs..etc.
➔ Implement the services:
➔ script folder : if the service can be implemented with xml.
➔ src folder : if the service to be implemented with java.

3) Presentation layer : display pages "user-interface"

➔ Decide what pages you want, what do you want to display to the user.
➔ Build this layer:
➔ widget folder : contains the screens that represent the application pages.
➔ webapp/"appfolder" folder : contains the forms that might be included in some screens.

4) Controller :

➔ It contains all the URLs related to the application, it receives the requests and forwards the requests to their location. It also defines the types of the requests, the handlers for the different events,…etc.

➔ contains the path for all the screens.

➔ Located in : webapp/WEB-INF/ controller.xml .
➔ When ever you add any new service or screen, you need to include it in the controller.

**[1] Taken from http://opensourcestrategies.com/ofbiz**
**[1]* Taken by word from http://opensourcestrategies.com/ofbiz**

# THE END