|  |
| --- |
| OFBIZ 초보자 개발 가이드 |
| SampleModule 개발 |
| Samuel Park |

[**개발 가이드 어플리케이션 생성** 1](#_Toc437518439)

[개요 1](#_Toc437518440)

[Prerequisite : 1](#_Toc437518441)

[해야 할 것과 하지 말아야 할 것 1](#_Toc437518442)

[Part 1 1](#_Toc437518443)

[컴포넌트 정의 파일 생성 1](#_Toc437518444)

[컴포넌트 정의 파일 ofbiz-component.xml 설명 1](#_Toc437518445)

[웹 어플리케이션 구성자 controller.xml, web.xml 생성 1](#_Toc437518446)

[사용자 인터페이스(UI : User Interface) 생성 1](#_Toc437518447)

[어플리케이션 리뷰 1](#_Toc437518448)

[Run the "SampleModule" Applicaction 1](#_Toc437518449)

[Part 2 1](#_Toc437518450)

[좀더 진보된 UI(User Interface) 만들기 1](#_Toc437518451)

[이제 브라우져 화면상의 Form에 Person엔티티 내용을 표시하는 방법을 설명한다. (Form Widget을 사용해서) 1](#_Toc437518452)

[이 어플리케이션의 화면 꾸미기를 위하여 “main Decorator” 생성 1](#_Toc437518453)

[SampleModule어플리케이션에서 바(app bar) 형태 메뉴 1](#_Toc437518454)

[UI 라벨 생성 1](#_Toc437518455)

[인증에 의한 어플리케이션 보안 1](#_Toc437518456)

[Part 3 1](#_Toc437518457)

[엔티티 CRUD(Create,Read,Update,Delete) 수행 코드 작성 1](#_Toc437518458)

[서비스를 작성 1](#_Toc437518459)

[부서 엔티티를 위한 CRUD수행 코드 작성 1](#_Toc437518460)

[사원 엔티티를 위한 CRUD수행 코드 작성 1](#_Toc437518461)

[이벤트 작성 1](#_Toc437518462)

[Simple Minilang Event 1](#_Toc437518463)

[Java Event 1](#_Toc437518464)

# **개발 가이드 어플리케이션 생성**

## 개요

가이드 문서는 6단계로 구분 되어 있음:

**1단계:**  이 단계에서는 사용자 정의 컴포넌트를 생성하고 실행하는 방법과 처음으로 화면에 “SampleModule개발 가이드”를 표시하는 방법을 배우게 됨.

**2단계:**  진보된 사용자 화면 구성 패턴을 생성해보고 화면과 데이터베이스 테이블 연계, 보안이 보장된 어플리케이션을 개발하는 방법을 알아봄.

**3단계:**  데이터베이스 객체와 상호작용하여 CRUD(Create, Retrieve, Update, Delete) 기능 구현, 필드 정합성 체크 이벤트 처리를 위한 서비스를 개발하는 것을 알아봄.

**4단계:**  이 단계에서는 사용자정의 조건하에서 자동적인 트리거 서비스를 부르는 방법을 배우고, 그룹과 같은 복합 서비스들을 부르며, 서비스들의 공통 파라미터들을 공유하고, 인터페이스 개념을 사용하는 방법이 설명 될 것이다.

**5단계:**  사용자 정의 객체를 생성하고, 기존의 OFBIZ객체를 확장하며, 사용자 정의 어플리케이션을 위한 XML 데이터를 만드는 법을 알 수 있음.

**6단계:**  마지막 단계로 클라이언트와 서버사이드 어플리케이션간 통신하는 웹서비스 개발을 위한 Apache Axis를 사용하는 방법을 익히고 나면 아파치 오픈소스 기반 OFBIZ프레임워크 활용한 SampleModule 개발자에게 필요한 기본 지식이 완성될 것임.

## Prerequisite :

1. OFBIZ버젼 : apache-ofbiz-13.07.02
2. JDK : jdk1.8.0\_66
3. Eclipse : Version: Mars.1 Release (4.5.1), Build id: 20150924-1200

* Eclipse 개발 환경 구성 시 OFBIZ최신 버전을 Repository로 부터 다운받아서 설치하면 일반적으로 프로젝트 설정 환경의 차이로 빨간색 느낌표가 나오는데 빌드 패쓰를 수정하여 정상적인 환경이 되도록 설정을 변경해 주어야 한다.

## 해야 할 것과 하지 말아야 할 것

1. 추가적인 프레임워크 또는 코드, 예제에 대한 의문이 있을 경우 Example component를 활용하고 거기에는 OFBIZ의 최신 버전의 내용이 반영 되어 있음.
2. 어플리케이션 개발전에 아래 내용을 읽어보기 바람 :
   1. [OFBiz Contributors Best Practice](https://cwiki.apache.org/confluence/display/OFBADMIN/OFBiz+Contributors+Best+Practices) ,
   2. [Coding Conventions](https://cwiki.apache.org/confluence/display/OFBADMIN/Coding+Conventions) and
   3. [Best Practice Guide](https://cwiki.apache.org/confluence/display/OFBADMIN/Best+Practices+Guide)
3. 다른 컴포넌트의 파일을 복사하지 말것, 파일 개정 버전번호도 함께 복사됨. 항상 새로운 파일을 생성하여 사용하기를 권장함. 만약의 경우 복사할 경우 문제 부분을 제거 후 복사하기 바람.
4. 좋은 적용사례를 찾기 위한 최적의 위치는 :  [OFBiz Documentation Index](https://cwiki.apache.org/confluence/display/OFBADMIN/OFBiz+Documentation+Index).
5. 콘솔 로그를 자주 읽어보는 것이 문제해결을 쉽게 하는 좋은 습관이 될 것이고 시스템 내부 프로세스 단계를 이해하는 데도 도움이 많이 됨.

# Part 1

## 컴포넌트 정의 파일 생성

1. **hot-deploy**디렉토리안에 “SampleModule”서브 디렉토리 생성.  
   Note : Remember that all development for this componentis done in this folder only.
2. **hot-deploy/SampleModule**경로상에 검포넌트 정의 파일인 **ofbiz-component.xml** 생성하고 주요 내용으로는 아래 예제1-1를 참조하고 프레임워크하의 기존에 다른 컴포넌트 정의 내용를 참조해서 필요 내용을 추가한다.

<예제1-1>

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <ofbiz-component name="SampleModule"         xmlns:xsi="[http://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance)         xsi:noNamespaceSchemaLocation="[http://ofbiz.apache.org/dtds/ofbiz-component.xsd"](http://ofbiz.apache.org/dtds/ofbiz-component.xsd)>        <resource-loader   name="main" type="component"/>      <webapp name="SampleModule"         title="SampleModule"         server="default-server"         base-permission="OFBTOOLS"         location="webapp/ SampleModule "         mount-point="/ SampleModule "         app-bar-display="false"/>  </ofbiz-component> |

## 컴포넌트 정의 파일 ofbiz-component.xml 설명

**ofbiz-component.xml** 파일은 컴포넌트 리소스 위치를 OFBIZ 프레임워크에게 알리는 역할을 하고 이것의 위치를 classpath에 추가한다.

<**resource-loader>** tag 의 **name**속성은 어떤 string이든지 가능하고 여기에는 "**main**"으로 명명하며, **type** 속성은 컴포넌트가 적재 될 것을 OFBiz에게 알린다.

|  |
| --- |
| <resource-loader name="main" type="component"/> |

**<webapp>** tag는 7가지 속성을 갖고 있다:

|  |
| --- |
| <webapp name="SampleModule"         title="The SampleModule Development Guide "         server="default-server"         base-permission="OFBTOOLS"         location="webapp/SampleModule"         mount-point="/SampleModule"         app-bar-display="false"/> |

 Their meanings are shown in the following table

| **속성** | **설명** | **필수여부** |
| --- | --- | --- |
| name | 웹 어플리케이션 이름 | true |
| title | 메뉴 네비게이션에 표시될 어플리케이션의 타이틀 | true |
| server | 사용할 서버 | true |
| base-permission | 이 컴포넌트 사용에 요구되는 권한으로 이 예제는 사용자에게 “OFBTOOLS” 권한이 요구 되어지며 'admin'유저는 이 컴포넌트를 테스트 하기 위한 어떤 신규 유저 생성 권한도 갖지 않는다. | true |
| location | 서버를 위한 기본 디렉토리 | true |
| mount-point | 리소스에 접근 하기 위한 URL로 이 경우에는 localhost:8080/SampleModule. | true |
| app-bar-display | common OFBiz decorator의 부분으로 메인 어플리케이션 메뉴 탭에보여 져야 할 경우에는 **true** 로 설정 | true |

## 웹 어플리케이션 구성자 controller.xml, web.xml 생성

생성 할 어플리케이션은 J2EE 웹어프리케이션 표준을 따른다.

1. “SampleModule” 컴포넌트 안에 "webapp" 디렉토리생성 (**hot-deploy/SampleModule/webapp**).  
   이 디렉토리는 우리가 생성할 컴포넌트의 웹 어플리케이션 관련된 모든 파일을 포함한다.
2. 개발할 웹 어플리케이션 이름이 "SampleModule"일때, **hot-deploy/SampleModule/webapp/SampleModule**디렉토리 생성하고, 컴포넌트는 다중의 웹 어플리케이션으로 구성 될 수 있는데 예로서 **"marketing"** 컴포넌트는 **"marketing"** 과 **"sfa"**라는 두 개의 웹 어플리케이션을 가질수 있다.
3. **hot-deploy/SampleModule/webapp/SampleModule/WEB-INF** 디렉토리 생성.  
   하나의 OFBiz web application은 **controller.xml** 과 **web.xml** 두 개의 구성자 파일을 요구한다. controller.xml 파일은 브라우져에서 사용자의 다양한 요청에 대해 무엇을 하고 어떠 액션을 취해서 무슨 페이지를 표시할 것인가를 OFBiz프레임워크에게 알리는 역할을 한다. web.xml은 OFBiz프레임워크에게 이 웹 어플리케이션을 위해 어떤 리소스(database and business logic access)를 이용하고 어떻게 관련된 이슈(로그인,리다이랙트,오류처리등..)를 취급 할 것인지를 알리는 역할을 한다.
4. 먼저 "web.xml"(web.xml : j2ee webapp specs)생성. 이 파일의 내용은 기존의 컴포넌트에서 자르기 붙이기 될 수 있으며,(예:/framework/example component) 변경될 주요 내용은 <display-name>테그 내용과와 **localDispatcherName**, webSiteId이다.

|  |
| --- |
| <context-param>      <param-name>webSiteId</param-name>      <param-value>SampleModule</param-value>      <description>A unique ID used to look up the WebSite entity to get information about catalogs, etc.</description>  </context-param>  <context-param>       <param-name>localDispatcherName</param-name>       <param-value>SampleModule</param-value>       <description>A unique name used to identify/recognize the local dispatcher for the Service Engine</description>  </context-param> |

1. 먼저websiteId의 값을 "SampleModule"로 설정하고.
2. mainDecoratorLocation은 값을 "component://SampleModule/widget/CommonScreens.xml"로 설정하고 위치는 화면에서 아래와 같이 main decorator location을 가리킨다.

|  |
| --- |
| ${parameters.mainDecoratorLocation} |

1. 이것은 main decorator location변경이 필요 할 때 여러 곳에서 path 변경으로 부터 독립성을 증가시킨다. 화면의 재사용성을 목적으로 활용됨.
2. "controller.xml" (used by ofbiz webapp controller)을 생성하고 이것은 처음에는 작고 간단하지만 향후에 기능이 추가됨으로써 복잡해 지게 된다. 우선 다음 코드를 추가 :

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <site-conf xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:noNamespaceSchemaLocation=*"http://ofbiz.apache.org/dtds/site-conf.xsd"*>  <include location=*"component://common/webcommon/WEB-INF/common-controller.xml"*/>  <description>SampleModule Component Site Configuration File</description>  <errorpage>/error/error.jsp</errorpage>  <owner>Copyright 2015 AIM Systems co ltd,.</owner>  <!-- Request Mappings -->  <request-map uri=*"main"*>  <security https=*"false"* auth=*"false"*/>  <response name=*"success"* type=*"view"* value=*"main"*/>  </request-map>  <!-- end of request mappings -->  <!-- View Mappings -->  <view-map name=*"main"* type=*"screen"* page=*"component://SampleModule/widget/SampleModuleScreens.xml#main"*/>  <!-- end of view mappings -->  </site-conf> |
|  |

## 사용자 인터페이스(UI : User Interface) 생성

1. hot-deploy/SampleModule/webapp/SampleModule/error디렉토리 생성.
2. "error" 디렉토리에 error.jsp파일 생성하고 이 파일의 내용은 기존 컴포넌트에서 자르기 붙이기로 추가하며(e.g. example component.), error page 위치는 처음에 <errorpage>/error/error.jsp</errorpage> 와 같이 controller.xml파일에 명시될 것이다. 사용자에게 에러 메시지가 보여지게 하기 위해서는 “~/webapp/SampleModule/error/” 디렉토리에 error.jsp page를 작성한다.
3. 컴포넌트 디렉토리 "SampleModule"에 "widget"(hot-deploy/SampleModule/widget)이라는 디렉토리를 생성한다. 이 디렉토리는 유저 인터페이스를 위해 사용할 forms, menus, 그리고 screens을 포함한다.
4. "widget" 디렉토리 안에 "SampleModuleScreens.xml"을 작성한다. 파일의 내용은 기존 컴포넌트(e.g. example component)에서 관련 내용을 자르기 붙이기 할 수 있으며, 여기서는 우선 아래 내용과 같이 작성한다.:

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <screens xmlns:xsi="[http://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance)       xsi:noNamespaceSchemaLocation="[http://ofbiz.apache.org/dtds/widget-screen.xsd"](http://ofbiz.apache.org/dtds/widget-screen.xsd)>      <screen name="main">          <section>              <widgets>                  <label text="This is first SampleModule"/>              </widgets>          </section>      </screen>  </screens> |

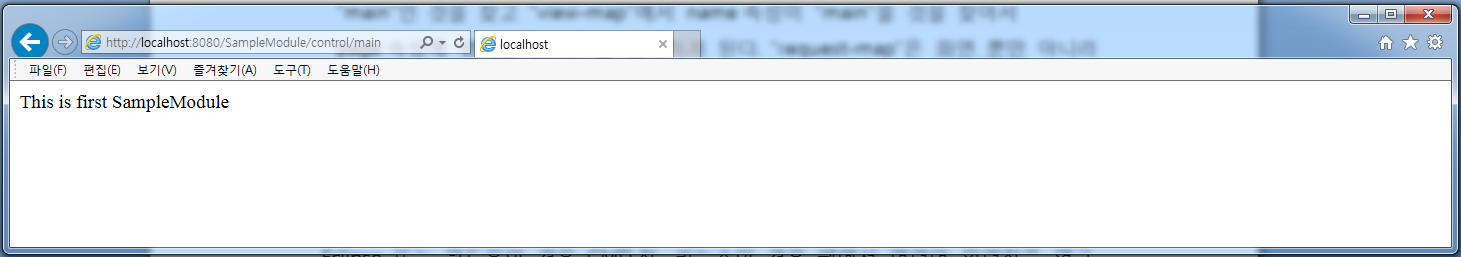
## 어플리케이션 리뷰

여기서 가장 기본적인 요소만으로 리뷰를 하여 이후 크고 복잡한 컴포넌트를 개발하는데 있어서도 문제가 없도록 기본적인 내용을 체크해 보며, 먼저 전체적인 흐름을 보기 위해 브라우저에서 다음과 같은 요청으로 시작한다 :"localhost:8080/SampleModule/control/main" *.*

1. OFBiz프레임워크가 "localhost:8080/SampleModule/control/main" 라는 요청을 만나면 먼저 /SampleModule를 가리키는데 이것은 ofbiz-component.xml파일 안에 웹 어플리케이션의 마운트 지점이 /SampleModule로 명시 되었기 때문이며 이후 프레임워크는 SampleModule라는 컴포넌트에서 요청된 구체적인 내용을 처리 한다.
2. 그러면 OFBiz프레임워크는 controller.xml 파일을 체크하게 되고 controller.xml파일 안에는 요청을 받는 ”request-maps”와 처리 결과를 표시하는 “view-maps”가 있으며, 만약 요청이 “main”이라는 이름이면 ”request-maps”의 uri속성 값이 “main”인 것을 찾고 “view-map”에서 name속성이 “main”을 것을 찾아서 page속성에 명시된 screen을 표시하게 된다. “request-map”은 화면 뿐만 아니라 나중에 알게 될 event와service에 대해서도 동일한 방법으로 사용된다.

## Run the "SampleModule" Applicaction

1. 자 이제 처음으로 SampleModule application을 실행 할 수 있게 되었다.

Eclipse 또는 윈도우의 경우 CMD창, 리눅스의 경우 쉘에서 명령어 입력창을 열고 ofbiz root디렉토리에서 ant start하여 서버를 실행하고 브라우져에서 다음 url [http://localhost:8080/SampleModule/control/main](http://localhost:8080/practice/control/main) 를 입력하면 브라우져에 "This is first SampleModule"을 표시한다.  
**Output Screen :**  


1. "SampleModule" 의 webapp/SampleModule 디렉토리에 index.jsp를 생성. (이 파일의 내용은 기존 다른 컴포넌트(e.g. example component )에서 복사) 이 파일은  [http://localhost:8080/SampleModule/](http://localhost:8080/practice/) 과 같이 URL이 주어지면 “control/main”으로 리다이렉트 한다. URL이  [http://localhost:8080/SampleModule/unknown/request](http://localhost:8080/practice/unknown/request) 로 주어지면 web.xml에 주어진 redirectPath로 리다이렉트 한다.

# Part 2

## 좀더 진보된 UI(User Interface) 만들기

1. 이제는 이 어플리케이션의 화면을 꾸미기 위해 "widget" 디렉토리에 CommonScreens.xml 파일을 생성한다. 이 파일은 SampleModule전체 어플리케이션 화면이 사용 할 공통화면을 포함 하게 될 것이다. 공통화면은 화면을 꾸미는데 사용되는 상단화면(header)과 아래단화면(footer)을 포함한다.

(예로서 자세한 내용은 기존 "example" 컴포넌트의 CommonScreens.xml을 참조. ) CommonScreens.xml의 내용은

|  |
| --- |
| <screen name="CommonSampleModuleDecorator">        <section>            <widgets>                <decorator-section-include name="body"/>            </widgets>        </section>  </screen> |

보다 자세한 사항은 “[Understanding the OFBiz Widget Toolkit](https://cwiki.apache.org/confluence/display/OFBIZ/Understanding+the+OFBiz+Widget+Toolkit)” 과  [FAQ](https://cwiki.apache.org/confluence/display/OFBIZ/FAQ+-+Tips+-+Tricks+-+Cookbook+-+HowTo)에서 "The Decorator" 를 읽어 보기 바랍니다.

web.xml에 CommonScreens.xml참조 추가

|  |
| --- |
| <context-param>       <param-name>commonDecoratorLocation</param-name>       <param-value>component://SampleModule/widget/CommonScreens.xml</param-value>       <description>The location of the common-decorator screen to use for this webapp; referred to as a context variable in screen def XML files.</description>  </context-param> |

1. 이 어플리케이션을 위한 메뉴를 표시하기 위하여 SampleModuleMenus.xml을 "widget"디렉토리에 만든다. ("example" 컴포넌트에서 ExampleMenus.xml 참조)

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <menus xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:noNamespaceSchemaLocation=*"http://ofbiz.apache.org/dtds/widget-menu.xsd"*>  <menu name=*"SampleModuleAppBar"* default-menu-item-name=*"main"* id=*"app-navigation"* type=*"simple"* title=*"SampleModule Application"*  default-selected-style=*"selected"* selected-menuitem-context-field-name=*"headerItem"*>  </menu>  </menus> |

이 SampleModuleMenus.xml파일은 CommonScreens.xml파일의 screen 엘리먼트에서 name속성의 값으로 “CommonSampleModuleDecorator”라는 이름을 갖는 screen엘리먼트 내의 include-menu 엘리먼트의 location 속성 값으로 아래와 같이 정의된다.

|  |
| --- |
| <screen name="CommonSampleModuleDecorator">          <section>              <widgets>                  <include-menu location="component://SampleModule/widget/SampleModuleMenus.xml" name="SampleModuleAppBar"/>                  <decorator-section-include name="body"/>              </widgets>          </section>  </screen> |

1. WEB-INF디렉토리 아래 "actions"라는 서브디렉토리를 생성하고 이 디렉토리에 스크립팅 파일을 작성 하려고 하며, 스크립팅 파일은 화면에서 사용될 데이터를 준비하고, 이 파일은 groovy파일 이라고 한다. 일찍이 bsh(beanshell)파일을 사용 했었고 UI에서 데이터베이스로부터 필요 데이터를 가져오는 스크립트로 사용 하였었다.

참조 : [Tips & Tricks while working with Groovy](https://cwiki.apache.org/confluence/pages/viewpage.action?pageId=6553850) & <http://groovy.codehaus.org/>.

"actions"디렉토리에 Person.groovy를 작성하고 이것은 "Person"엔티티에서 모든 레코드를 가져올것이다. 여기서는 아래와 같이 단순한 코드가 추가 될 것이다.

|  |
| --- |
| context.persons = delegator.findList("Person", null, null, null, null, false); |

위의 문장은 "Person"이라는 엔티티에서 모든 레코드를 가져 오고 “persons”라는 이름의 건텍스트 객체에 내용을 적재 할 것이다. 그리고 이름이 person이라는 list가 .ftl파일안에서 레코드를 보여주기 위해 반복 처리된다.

화면에서 자주 참조되는 변수에 대해서는 다음을 참조하기 바랍니다. : [Which variables are available in screen context?](https://cwiki.apache.org/confluence/pages/viewpage.action?pageId=6553757)

1. 이제는 webapp디렉토리 아래의 "SampleModule" 디렉토리에 Person.groovy파일에서 가져온 데이터를 보여주기 위해 "Person.ftl"라는 ftl파일을 작성한다.

자세한 것은 다음 참조 : <http://freemarker.sourceforge.net/docs/>  
"Person.ftl" 코딩 내용은 다음 같다 :

|  |
| --- |
| <#if persons?has\_content>    <h2>Some of the people who visited our site are:</h2>    <br>    <ul>      <#list persons as person>        <li>${person.firstName?if\_exists} ${person.lastName?if\_exists}</li>      </#list>    </ul>  </#if> |

1. SampleModuleScreens.xml파일의 screen 엘리먼트에 name속성의 값으로 "person"이라는 새로운 screen엘리먼트를 추가하고 또한 SampleModuleMenus.xml파일에 새로운 menu항목을 추가한다.

SampleModuleScreens.xml :

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <screens xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:noNamespaceSchemaLocation=*"http://ofbiz.apache.org/dtds/widget-screen.xsd"*>  <screen name=*"main"*>  <section>  <widgets>  <label text=*"This is first SampleModule"*/>  </widgets>  </section>  </screen>  <screen name=*"person"*>  <section>  <actions>  <script location=*"component://SampleModule/webapp/SampleModule/WEB-INF/actions/Person.groovy"*/>  </actions>  <widgets>  <decorator-screen name=*"CommonSampleModuleDecorator"* location=*"${parameters.commonDecoratorLocation}"*>  <decorator-section name=*"body"*>  <platform-specific>  <html>  <html-template location=*"component://SampleModule/webapp/SampleModule/Person.ftl"*/>  </html>  </platform-specific>  </decorator-section>  </decorator-screen>  </widgets>  </section>  </screen>  </screens> |

SampleModuleMenus.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <menus xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:noNamespaceSchemaLocation=*"http://ofbiz.apache.org/dtds/widget-menu.xsd"*>  <menu name=*"SampleModuleAppBar"* default-menu-item-name=*"main"* id=*"app-navigation"* type=*"simple"* title=*"SampleModule Application"*  default-selected-style=*"selected"* selected-menuitem-context-field-name=*"headerItem"*>  <menu-item name=*"main"* title=*"Main"*><link target=*"main"*/></menu-item>  <menu-item name=*"person"* title=*"Person"*><link target=*"person"*/></menu-item>  </menu>  </menus> |

1. controller.xml에 “person” 관련 새로운 request-map과 view-map 엘리먼트를 추가한다.

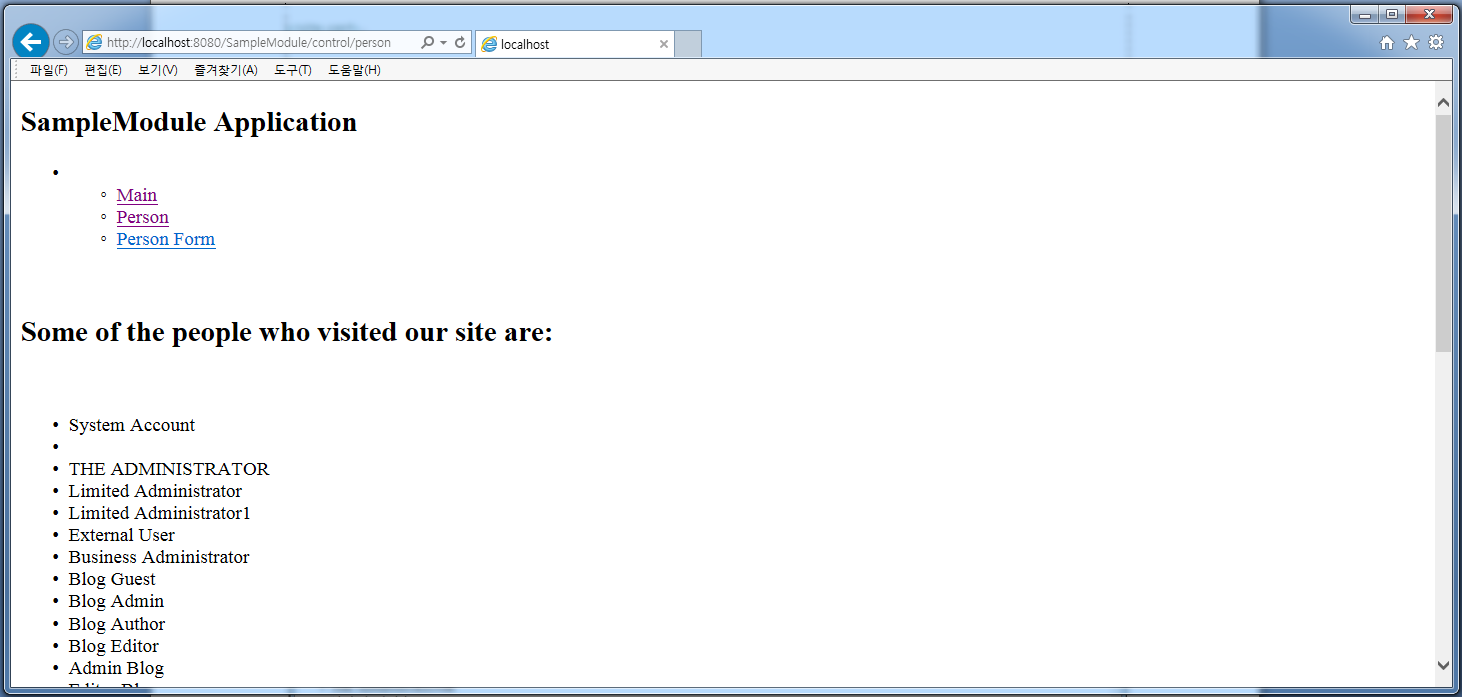
controller.xml 파일은 다음과 같다.:

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <site-conf xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:noNamespaceSchemaLocation=*"http://ofbiz.apache.org/dtds/site-conf.xsd"*>  <include location=*"component://common/webcommon/WEB-INF/common-controller.xml"*/>  <description>SampleModule Component Site Configuration File</description>  <owner>Copyright 2001-2009 The Apache Software Foundation</owner>  <handler name=*"screen"* type=*"view"* class=*"org.ofbiz.widget.screen.MacroScreenViewHandler"*/>  <!-- Request Mappings -->  <request-map uri=*"main"*>  <security https=*"false"* auth=*"false"*/>  <response name=*"success"* type=*"view"* value=*"main"*/>  </request-map>  <request-map uri=*"person"*>  <security auth=*"false"* https=*"false"*/>  <response name=*"success"* type=*"view"* value=*"person"*/>  </request-map>  <!-- end of request mappings -->  <!-- View Mappings -->  <view-map name=*"main"* type=*"screen"* page=*"component://SampleModule/widget/SampleModuleScreens.xml#main"*/>  <view-map name=*"person"* type=*"screen"* page=*"component://SampleModule/widget/SampleModuleScreens.xml#Person"*/>  <!-- end of view mappings -->  </site-conf> |

이제 다시 브라우져에 아래 URL을 입력하고 실행한다. [http://localhost:8080/SampleModule/control/person](http://localhost:8080/aimOS1guide/control/person) .

힌트

만약 브라우져의 출력된 화면이 메뉴를 포함하고 있지 않으면 controller.xml 파일안의 auth="false"를 auth="true"로 변경하는 것이 필요하다

**Output Screen :**  


## 이제 브라우져 화면상의 Form에 Person엔티티 내용을 표시하는 방법을 설명한다. (Form Widget을 사용해서)

1. 먼저 SampleModuleMenus.xml파일의 menu엘리먼트 아래 menu-item 엘리먼트 name속성값이 "personForm"인 menu-item 엘리먼트를 추가함.

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <menus xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:noNamespaceSchemaLocation=*"http://ofbiz.apache.org/dtds/widget-menu.xsd"*>  <menu name=*"SampleModuleAppBar"* default-menu-item-name=*"main"* id=*"app-navigation"* type=*"simple"* title=*"SampleModule Application"*  default-selected-style=*"selected"* selected-menuitem-context-field-name=*"headerItem"*>  <menu-item name=*"main"* title=*"Main"*><link target=*"main"*/></menu-item>  <menu-item name=*"person"* title=*"Person"*><link target=*"person"*/></menu-item>  <menu-item name=*"personForm"* title=*"Person Form"*><link target=*"personForm"*/></menu-item>  </menu>  </menus> |

1. “widget”디렉토리에 이름이 SampleModuleForms.xml인 파일을 추가하고 내용으로 아래와 같이 Person엔티티로부터 레코드의 내용을 보이기 위해 form엘리먼트의 각각의 내용을 정의하여 추가한다.(자세한 내용은 ExampleScreens.xml과 ExampleForms.xml 파일을 참조)

|  |
| --- |
| <form name="ListPersons" type="list" list-name="persons" list-entry-name="person"  default-map-name="person" paginate-target="personForm">       <!-- Important: Here service definition for updateSampleModulePerson has been used for automatically rendering the form fields, which you can use after completing CRUD operations from Part-3 -->       <!-- auto-fields-service service-name="updateSampleModulePerson" default-field-type="display" map-name="person"/-->         <!-- The above method can be used in case a service specific form is being rendered, otherwise form-fields can be explicitly mentioned as given below:-->       <field name="firstName"><display/></field>       <field name="middleName" ><display/> </field>       <field name="lastName" ><display/> </field>  </form> |

1. SampleModuleScreens.xml파일의 screen 엘리먼트에 name속성의 값으로 "personForm"이라는 새로운 screen 엘리먼트를 추가.

|  |
| --- |
| <screen name="personForm">          <section>              <actions>                  <set field="headerItem" value="personForm"/>                  <set field="titleProperty" value="PageTitleSampleModulePersonForm"/>                  <entity-condition entity-name="Person" list="persons"/>              </actions>              <widgets>                  <decorator-screen name="CommonSampleModuleDecorator" location="${parameters.commonDecoratorLocation }">                      <decorator-section name="body">                          <label text="Person List" style="h2"/>                          <include-form name="ListPersons" location="component://SampleModule/widget/SampleModuleForms.xml"></include-form>                      </decorator-section>                  </decorator-screen>              </widgets>          </section>  </screen> |

1. controller.xml에 “personForm” 관련 새로운 request-map과 view-map 엘리먼트를 추가한다.

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <site-conf xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:noNamespaceSchemaLocation=*"http://ofbiz.apache.org/dtds/site-conf.xsd"*>  <include location=*"component://common/webcommon/WEB-INF/common-controller.xml"*/>  <description>SampleModule Component Site Configuration File</description>  <owner>Copyright 2001-2009 The Apache Software Foundation</owner>  <handler name=*"screen"* type=*"view"* class=*"org.ofbiz.widget.screen.MacroScreenViewHandler"*/>  <!-- Request Mappings -->  <request-map uri=*"main"*>  <security https=*"false"* auth=*"false"*/>  <response name=*"success"* type=*"view"* value=*"main"*/>  </request-map>  <request-map uri=*"person"*>  <security auth=*"false"* https=*"false"*/>  <response name=*"success"* type=*"view"* value=*"person"*/>  </request-map>  <request-map uri=*"personForm"*>  <security https=*"false"* auth=*"false"*/>  <response name=*"success"* type=*"view"* value=*"personForm"*/>  </request-map>  <!-- end of request mappings -->  <!-- View Mappings -->  <view-map name=*"main"* type=*"screen"* page=*"component://SampleModule/widget/SampleModuleScreens.xml#main"*/>  <view-map name=*"person"* type=*"screen"* page=*"component://SampleModule/widget/SampleModuleScreens.xml#person"*/>  <view-map name=*"personForm"* type=*"screen"* page=*"component://SampleModule/widget/SampleModuleScreens.xml#personForm"*/>  <!-- end of view mappings -->  </site-conf> |

여기서 어플리케이션을 다시 실행해 보고 차이를 비교해 보기 바람.  
Till Now you have worked on controller requests mappings, Screen widget, form widget, Decorator, Menus, groovy, ftl.

## 이 어플리케이션의 화면 꾸미기를 위하여 “main Decorator” 생성

1. SampleModule컴포넌트 "widget"(hot-deploy/SampleModule/widget)디렉토리 안의 CommonScreens.xml파일에 name속성의 값으로 "main-decorator"로 정의되는 screen엘리먼트를 추가.(Example 컴포넌트의 CommonScreens.xml 참조)

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <screens xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:noNamespaceSchemaLocation=*"http://ofbiz.apache.org/dtds/widget-screen.xsd"*>  <screen name=*"CommonSampleModuleDecorator"*>  <section>  <widgets>  <include-menu location=*"component://SampleModule/widget/SampleModuleMenus.xml"* name=*"SampleModuleAppBar"*/>  <decorator-section-include name=*"body"*/>  </widgets>  </section>  </screen>  <screen name=*"main-decorator"*>  <section>  <actions>  <property-map resource=*"CommonUiLabels"* map-name=*"uiLabelMap"* global=*"true"*/>  <property-map resource=*"SampleModuleUiLabels"* map-name=*"uiLabelMap"* global=*"true"*/>  <set field=*"layoutSettings.companyName"* from-field=*"uiLabelMap.SampleModuleCompanyName"* global=*"true"*/>  <set field=*"activeApp"* value=*"SampleModule"* global=*"true"*/>  <set field=*"applicationMenuName"* value=*"SampleModuleAppBar"* global=*"true"*/>  <set field=*"applicationMenuLocation"* value=*"component://SampleModule/widget/SampleModuleMenus.xml"* global=*"true"*/>  </actions>  <widgets>  <include-screen name=*"GlobalDecorator"* location=*"component://common/widget/CommonScreens.xml"*/>  </widgets>  </section>  </screen>  </screens> |

1. web.xml에 context-param 엘리먼트의 param-name 엘리먼트 값을 mainDecoratorLocation로 설정하고 param-value 엘리먼트 값으로 프레임워크 공용 screen정의 파일인 CommonScreens.xml 참조 추가

|  |
| --- |
| <context-param>       <param-name>mainDecoratorLocation</param-name>       <param-value>component://SampleModule/widget/CommonScreens.xml</param-value>       <description>The location of the common-decorator screen to use for this webapp; referred to as a context variable in screen def XML files.</description>  </context-param> |

1. SampleModuleScreens.xml파일의 name속성의 값으로 "personForm"인 screen 엘리먼트에 widgets 엘리먼트 하위 엘리먼트인decorator-screen의 name 속성값으로 *CommonSampleModuleDecorator를 main-decorator로*, location속성 값으로 *"${parameters.commonDecoratorLocation}"*을 *"${parameters.mainDecoratorLocation}"*로 변경.

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <screens xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:noNamespaceSchemaLocation=*"http://ofbiz.apache.org/dtds/widget-screen.xsd"*>  <screen name=*"main"*>  <section>  <widgets>  <label text=*"This is first SampleModule"*/>  </widgets>  </section>  </screen>  <screen name=*"person"*>  <section>  <actions>  <script location=*"component://SampleModule/webapp/SampleModule/WEB-INF/actions/Person.groovy"*/>  </actions>  <widgets>  <decorator-screen name=*"CommonSampleModuleDecorator"* location=*"${parameters.commonDecoratorLocation}"*>  <decorator-section name=*"body"*>  <platform-specific>  <html>  <html-template location=*"component://SampleModule/webapp/SampleModule/Person.ftl"*/>  </html>  </platform-specific>  </decorator-section>  </decorator-screen>  </widgets>  </section>  </screen>  <screen name=*"personForm"*>  <section>  <actions>  <set field=*"headerItem"* value=*"personForm"*/>  <set field=*"titleProperty"* value=*"PageTitleSampleModulePersonForm"*/>  <entity-condition entity-name=*"Person"* list=*"persons"*/>  </actions>  <widgets>  <decorator-screen name=*"main-decorator"* location=*"${parameters.mainDecoratorLocation}"*>  <decorator-section name=*"body"*>  <label text=*"Person List"* style=*"h2"*/>  <include-form name=*"ListPersons"* location=*"component://SampleModule/widget/SampleModuleForms.xml"*></include-form>  </decorator-section>  </decorator-screen>  </widgets>  </section>  </screen>  </screens> |

1. 이제 *main-decorator* 라는 이름의 screen decorator가 추가되었다.(자세한 내용은 위 Part 2의 **좀더 진보된 UI(User Interface) 만들기** 참조)   
   다시 실행해 보고 차이를 확인하기 바람.

## SampleModule어플리케이션에서 바(app bar) 형태 메뉴

* 이것을 위해 ofbiz-component.xml 파일에서 app-bar-display="true"로 설정, 서버를 다시 시작 후 SampleModule어플리케이션에서 바 메뉴(app bar)를 볼 수 있다.

## UI 라벨 생성

1. UI라벨을 만들기 위해 컴포넌트(SampleModule) 디렉토리 아래 "config"라는 이름의 디렉토리 생성(SampleModule/config)   
   **주의: -** ofbiz-component.xml 파일 안에 config 디렉토리를 설정 해 주어야 하는것을 상기하기 바람.  
   그것은 다음과 같게 될것이다:

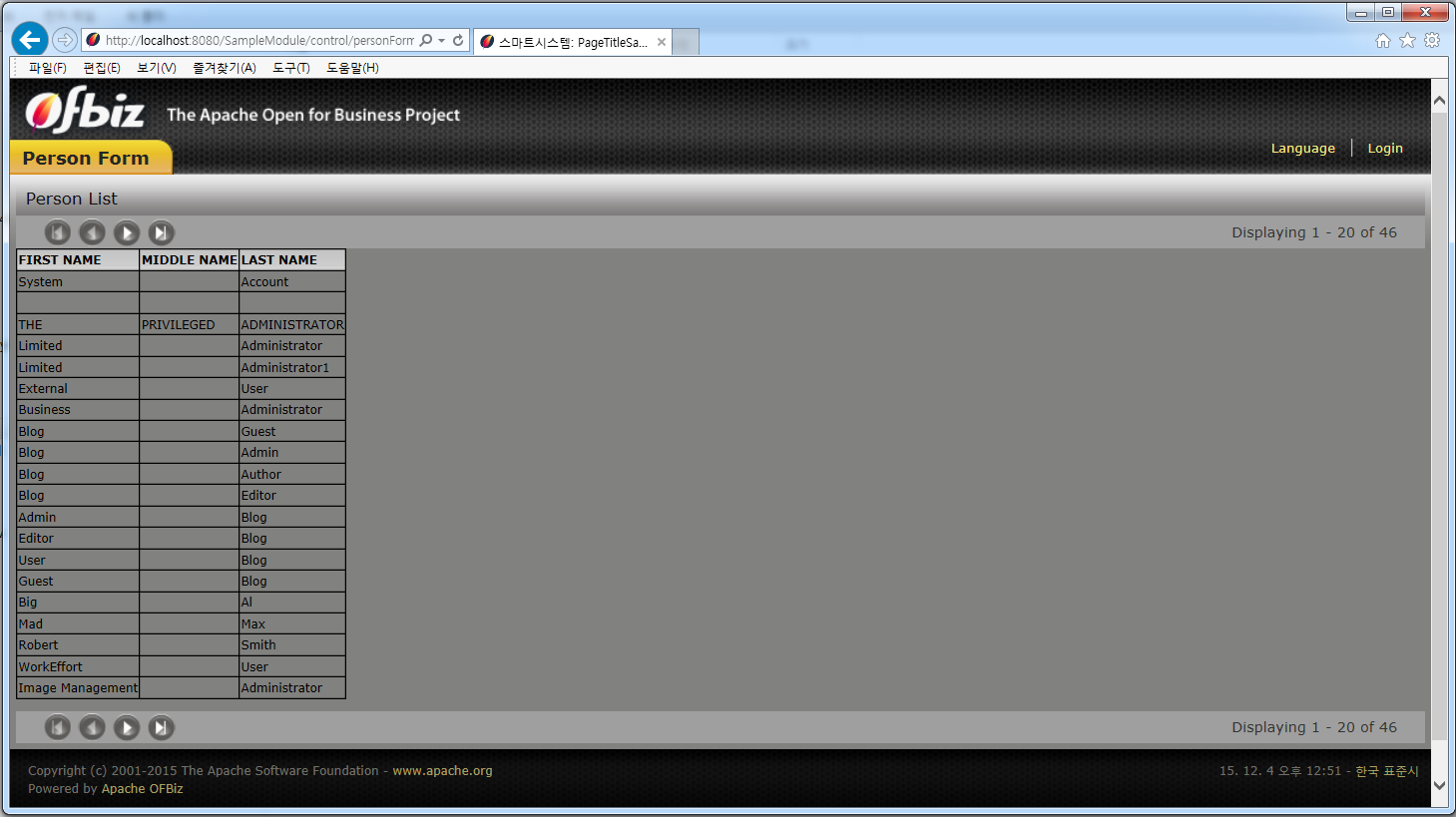
|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <ofbiz-component name=*"SampleModule"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:noNamespaceSchemaLocation=*"http://ofbiz.apache.org/dtds/ofbiz-component.xsd"*>  <resource-loader name=*"main"* type=*"component"*/>  <!-- place the config directory on the classpath to access configuration files -->  <classpath type=*"dir"* location=*"config"*/>  <webapp name=*"SampleModule"*  title=*"SampleModule Developement Guide"*  server=*"default-server"*  base-permission=*"OFBTOOLS"*  location=*"webapp/SampleModule"*  mount-point=*"/SampleModule"*  app-bar-display=*"true"*/>  </ofbiz-component> |

이상의 내용은 UI 라벨정의와 같은 Configuration 파일에 접근하기 위해 classpath가 config 디렉토리 정보를 인식하게끔 한다.

1. 그리고 config디랙토리에 SampleModuleUiLabels.xml라는 파일을 생성하고 SampleModule 어플리케이션을 위한 몇 개의 ui 라벨을 추가 한다.( ExampleUiLabels.xml을 참조) 항상 기억해야 할 것은 ui을 변경 추가시 언제든지 서버를 재 시작 해야지 어플리케이션에 반영된다는 것이다. 우선적으로 2개의 ui 라벨을 아래와 같이 추가한다

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <resource xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:noNamespaceSchemaLocation=*"http://ofbiz.apache.org/dtds/ofbiz-properties.xsd"*>  <property key=*"SampleModuleApplication"*>  <value xml:lang=*"en"*>This is first SampleModule</value>  <value xml:lang=*"fr"*>Application exemple</value>  <value xml:lang=*"it"*>Applicazione Esempi</value>  <value xml:lang=*"ja"*>サンプルアプリケーション</value>  <value xml:lang=*"ro"*>Aplicatie Exemple</value>  <value xml:lang=*"ru"*>Приложение Пример</value>  <value xml:lang=*"th"*>ตัวอย่างโปรแกรมคอมพิวเตอร์</value>  <value xml:lang=*"zh"*>样例应用</value>  <value xml:lang=*"zh-TW"*>範例應用</value>  <value xml:lang=*"ko\_KR"*>이것은 첫번째 SampleModule 어플리케이션입니다</value>  </property>  <property key=*"SampleModuleCompanyName"*>  <value xml:lang=*"en"*>AIM Systems: SampleModule</value>  <value xml:lang=*"ko\_KR"*>에임시스템</value>  </property>  </resource> |

1. 이 ui라벨은 이전에 작성한 SampleModule컴포넌트 "widget"(hot-deploy/SampleModule/widget)디렉토리 안의 CommonScreens.xml파일에 main decorator screen에 포함되어 있고 적당한 곳에서 이러한 2개의 ui라벨을 사용하기 바람.

**주의**: 항상 ofbiz프레임워크 안에서 사용하고자 하는 UI라벨을 먼저 찾아보고 없을 경우에만 새로운 것을 추가하기 바람.   
자 이제 어플리케이션을 실행해 보고 아래와 같은 결과 화면을 체크해 보기 바람:  
**Output Screen:**  


## 인증에 의한 어플리케이션 보안

1. 메뉴에 로그인/로그아웃관련 옵션 설정은 ExampleMenus.xml를 참조하기 바랍니다.   
   이러한 옵션의 이용은 어플리케이션 내의 controller.xml에서 include 엘리먼트의 location속성의 값으로 "component://common/webcommon/WEB-INF/common-controller.xml"을 설정하거나 아래와 같은 내용을 controller.xml에 추가한다.

|  |
| --- |
| <!- Request Mappings ->  <!-- Security Mappings -->   <request-map uri="checkLogin" edit="false">      <description>Verify a user is logged in.</description>          <security https="true" auth="false"/>          <event type="java" path="org.ofbiz.webapp.control.LoginWorker" invoke="checkLogin" />          <response name="success" type="view" value="main"/>          <response name="error" type="view" value="login"/>      </request-map>      <request-map uri="login">          <security https="true" auth="false"/>          <event type="java" path="org.ofbiz.webapp.control.LoginWorker" invoke="login"/>          <response name="success" type="view" value="main"/>          <response name="error" type="view" value="login"/>      </request-map>      <request-map uri="logout">          <security https="true" auth="true"/>          <event type="java" path="org.ofbiz.webapp.control.LoginWorker" invoke="logout"/>          <response name="success" type="request" value="checkLogin"/>          <response name="error" type="view" value="main"/>      </request-map> |

이상의 request-map들이 컴포넌트 내의 어떠한 controller.xml 파일에도 없다면 추가하는 것이 필요하고 만약 이미 common-controller.xml 파일이 어떤 controller.xml에서 include 엘리먼트의 location속성의 값으로 선언 되어 있다면 controller.xml 파일에 명시적으로 추가 기술 할 필요는 없다.  
마찬가지로 view-map으로 정의 되어 표시되는 로그인 화면도 request-map의 설명에서와 같이 공통으로 common-controller.xml 파일에 정의된 내용을 사용하거나 명시적으로 해당 어플리케이션의 controller.xml 파일에 아래와 같은 내용을 추가하여 적용 할 수 있다 :

|  |
| --- |
| <view-map name="login" type="screen" page="component://common/widget/CommonScreens.xml#login"/> |

1. controller.xml파일 안의 request-map 엘리먼트 아래의 security엘리먼트의 auth속성이 true(auth="true")로 설정 되면 이 requests는 인증이 필요하다는 것을 의미한다.

이러한 것은 1차적으로 적용된 보안수준이라고 볼 수 있고 아래와 같다 :

|  |
| --- |
| <request-map uri="main">      <security https="true" auth="true"/>      <response name="success" type="view" value="main"/>      <response name="error" type="view" value="main"/>  </request-map> |

이제 어플리케이션을 실행 해 보고 차이를 관찰하기 바라고 로그인 화면이 나타나게 되면 user name은 “admin” 패스워드는 “ofbiz”로 로그인 할 수 있다.

권한과 인증관련 구체적인 내용에 대해서는 다음 내용을 읽어보기 바랍니다 : [OFBiz Security](https://cwiki.apache.org/confluence/display/OFBTECH/OFBiz+security)

# Part 3

## 엔티티 CRUD(Create,Read,Update,Delete) 수행 코드 작성

엔티티(객체 또는 데이터베이스 테이블)의 CRUD수행 기능은 minilang으로 작성한 서비스에 의해 실행 될 것이고, CRUD수행 기능의 이해를 돕기 위해 먼저 서비스를 작성해 보는 것으로부터 시작하게 될 것이다. 그래서 먼저 이미 구현된 서비스를 불러서 실행해 보는 것으로 시작한다. 대상 엔티티로는 부서(조직)와 사원(직원) 엔티티를 활용한다

* + 부서(Party)
  + 사원(Person)

사원은 부서의 부서원(부서의 일부분)이다 그래서 사원을 생성하기 위해 먼저 부서 레코드중 부서 타입 식별자(PARTY\_TYPE\_ID) 필드의 데이타가 PERSON(partyTypeId="PERSON")으로 생성된 레코드가 필요하다. 사원 생성은 두 단계의 절차를 거쳐서 진행된다:

1. 부서 엔티티중 부서타입(party type)이 사원(PERSON)이라는 부서 레코드 생성을 위한 서비스를 작성
2. 사원을 추가하는 서비스에서 부서 레코드를 생성하는 서비스를 부르면서 사원 레코드를 생성하는 서비스를 작성

## 서비스를 작성

1. "SampleModule" 컴포넌트 디렉토리 아래에 먼저 "servicedef"이라는 디렉토리를 생성한다. 그리고 여기에는 모든 서비스 정의 파일들이 위치한다.( e.g. services.xml, secas.xml등..)

**주의**: 만약 서비스가 Java로 작성되었다면 그것은 "src"디렉토리에 위치하게 될 것이고(e.g. for java applications/party/src/org/ofbiz/party/party/PartyServices.java) 서비스가 minilang으로 작성되었다면 script디렉토리에 위치(for minilang applications/party/script/org/ofbiz/party/party/PartyInvitationServices.xml.) 할 것이다. 각각의 클래스 패쓰(class path)와 파일 패쓰(file path)가 서비스 정의서에 언급 되어져야 한다.

1. controller.xml에 서비스를 실행할 아래와 같이 request-map 엘리먼트를 추가하고 response도 설정한다.

|  |
| --- |
| ………………….  <request-map uri=*"createSampleModulePerson"*>  <security https=*"true"* auth=*"true"*/>  <event type=*"service"* invoke=*"createSampleModulePerson"*/>  <response name=*"success"* type=*"view"* value=*"personForm"*/>  </request-map>  …………………………. |

1. 작성된 모든 서비스는 서버가 시작할 때 적재 되어야 하고 아래와 같이 ofbiz-component.xml파일에 서비스 정의 파일인 services.xml 이 service-resource 엘리먼트를 이용하여 설정 되어야 한다.

|  |
| --- |
| ………………………………  <service-resource loader=*"main"* location=*"servicedef/services.xml"* type=*"model"* />  ………………………. |

1. 따라서 서비스 정의가 수정 될 때마다 변경을 반영하기 위해 반듯이 서버는 재 시작 되어야 한다.

## 부서 엔티티를 위한 CRUD수행 코드 작성

먼저 부서 엔티티을 위한 서비스를 작성하게 될 것이고 이것은 사원 엔티티 생성시 부서 엔티티 생성 서비스를 부르는 것이 필요한데 이때 사용하게 될 것이다.

1. servicedef디렉토리에 "services.xml"이라는 이름의 파일을 작성하고

부서 엔티티에 대한 CRUD수행을 위한 서비스를 정의하는데 service엘리먼트의 name속성에 이름이 createSampleModuleParty, updateSampleModuleParty, deleteSampleModuleParty인 service엘리먼트들을 추가하고 각각의 location속성에 ~SampleModule/script/org/hotwax/SampleModule/SampleModuleServices.xml 와 같이 서비스가 구현될 정확한 파일 위치를 기술한다.

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <services xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:noNamespaceSchemaLocation="http://ofbiz.apache.org/dtds/services.xsd">  <description>SampleModule Services</description>  <!-- Party related Services -->  <service name="createSampleModuleParty" default-entity-name="Party" engine="simple"  location="component://SampleModule/script/org/hotwax/SampleModule/SampleModuleServices.xml" invoke="createSampleModuleParty" auth="true">  <description>Create a Party</description>  <auto-attributes include="pk" mode="OUT" optional="false"/>  <auto-attributes include="nonpk" mode="IN" optional="true"/>  </service>  <service name="updateSampleModuleParty" default-entity-name="Party" engine="simple"  location="component://SampleModule/script/org/hotwax/SampleModule/SampleModuleServices.xml" invoke="updateSampleModuleParty" auth="true">  <description>Update a Party</description>  <auto-attributes include="pk" mode="IN" optional="false"/>  <auto-attributes include="nonpk" mode="IN" optional="true"/>  </service>  <service name="deleteSampleModuleParty" default-entity-name="Party" engine="simple"  location="component://SampleModule/script/org/hotwax/SampleModule/SampleModuleServices.xml" invoke="deleteSampleModuleParty" auth="true">  <description>Delete a Party</description>  <auto-attributes include="pk" mode="IN" optional="false"/>  </service>  ………………………. |

1. 컴포넌트 디렉토리에 이러한 서비스들이 위치하여 실행될 디렉토리 구조(SampleModule/script/org/hotwax/SampleModule/)와 SampleModuleServices.xml 파일을 생성(Example 컴포넌트에서 services.xml과 ExampleServices.xml참조), 여기서는 다음과 같이 생성함 :

~SampleModule/script/org/hotwax/SampleModule/SampleModuleServices.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <simple-methods xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:noNamespaceSchemaLocation=*"http://ofbiz.apache.org/dtds/simple-methods.xsd"*>  <!-- Party related Services -->  <simple-method method-name=*"createSampleModuleParty"* short-description=*"create a Party"*>  <make-value entity-name=*"Party"* value-field=*"newEntity"*/>  <sequenced-id sequence-name=*"Party"* field=*"newEntity.partyId"*/>  <!-- get the next sequenced ID -->  <log level=*"info"* message=*"==========PartyId======${newEntity.partyId}"*/>  <field-to-result field=*"newEntity.partyId"* result-name=*"partyId"*/>  <set-nonpk-fields map=*"parameters"* value-field=*"newEntity"*/>  <create-value value-field=*"newEntity"*/>  </simple-method>  ………………………. |

**중요**

1. Do not use the <override> tag as it is introduced later in the tutorial.  
   From this place if you want to run these services then you can run them by webtools--> Run Service . By this place you can test your services.
2. At this place you must read <http://markmail.org/message/dj4wvtm4u2nxoz3r>. This feature has been added against the traditional approach of writing CRUD operations for an entity.

이러한 새로운 특징은 수행되기 원하는 기능이 단지 선언 하는 것 만으로도 구현 될 수 있다는 것이다. 기본적으로 아래의 example 컴포넌트의 services.xml에서 보는 바와 같이 엔진 속성(the engine attribute)은 "entity-auto"로, 인보크 속성(the invoke attribute)은 "create", "update", 또는 "delete"로 설정 된다. :

|  |
| --- |
| <service name="createExample" default-entity-name="Example" engine="entity-auto" invoke="create" auth="true">      <description>Create a Example</description>      <permission-service service-name="exampleGenericPermission" main-action="CREATE"/>      <auto-attributes include="pk" mode="OUT" optional="false"/>      <auto-attributes include="nonpk" mode="IN" optional="true"/>      <override name="exampleTypeId" optional="false"/>      <override name="statusId" optional="false"/>      <override name="exampleName" optional="false"/>  </service> |

engine="entity-auto" invoke="create"로 설정은 "Example"이라는 기본 엔티티(default-entity)의 레코드를 생성하게 하는 역할임.

## 사원 엔티티를 위한 CRUD수행 코드 작성

- 사원 엔티티의 레코드 생성을 위해서는 부서 타입 식별자(PARTY\_TYPE\_ID, partyId)가 필요하고, 그래서 먼저 createSampleModuleParty 서비스를 불러 partyId를 얻어서 사원 레코드를 생성 하게 될 것이다.

- 사원정보를 입력 받기 위해 목록 화면 아래에 정보 추가 입력폼을 만들고 이 입력폼은 사원정보를 생성하는 서비스를 부르게 될 것이다.

1. 먼저 기존 ~SampleModule/widget/SampleModuleForms.xml파일에서 사원정보 생성을 위해 추가정보 입력화면을 만들 것인데 이를 위해 form 엘리먼트를 추가하고 name속성의 값을 *"CreatePerson"* 이라 하고 아래와 같이 기타 속성값을 정의해 준다.

|  |
| --- |
| <form name=*"CreatePerson"* type=*"single"* target=*"createSampleModulePerson"*>  <auto-fields-service service-name=*"createSampleModulePerson"*/>  <field name=*"submitButton"* title=*"Create"* widget-style=*"smallSubmit"*><submit button-type=*"button"*/></field>  </form> |

1. 화면을 정의한 이후 사원 엔티티를 위한 CRUD수행 코드를 작성하는데 이것은 ~SampleModule/servicedef/services.xml파일 안에 service 엘리먼트의 createSampleModulePerson이라는 name속성의 값으로 정의하며 아래와 같이 작성된다.

|  |
| --- |
| <!-- Person related Services -->  <service name=*"createSampleModulePerson"* default-entity-name=*"Person"* engine=*"simple"*  location=*"component://SampleModule/script/org/hotwax/SampleModule/SampleModuleServices.xml"* invoke=*"createSampleModulePerson"* auth=*"true"*>  <description>Create a Person</description>  <auto-attributes include=*"pk"* mode=*"OUT"* optional=*"false"*/>  <attribute name=*"salutation"* mode=*"IN"* type=*"String"* optional=*"true"*/>  <attribute name=*"firstName"* mode=*"IN"* type=*"String"* optional=*"false"*/>  <attribute name=*"middleName"* mode=*"IN"* type=*"String"* optional=*"true"*/>  <attribute name=*"lastName"* mode=*"IN"* type=*"String"* optional=*"false"*/>  <attribute name=*"suffix"* mode=*"IN"* type=*"String"* optional=*"true"*/>  </service> |

1. Update, Delete도 services.xml파일안에 유사한 방법으로 추가

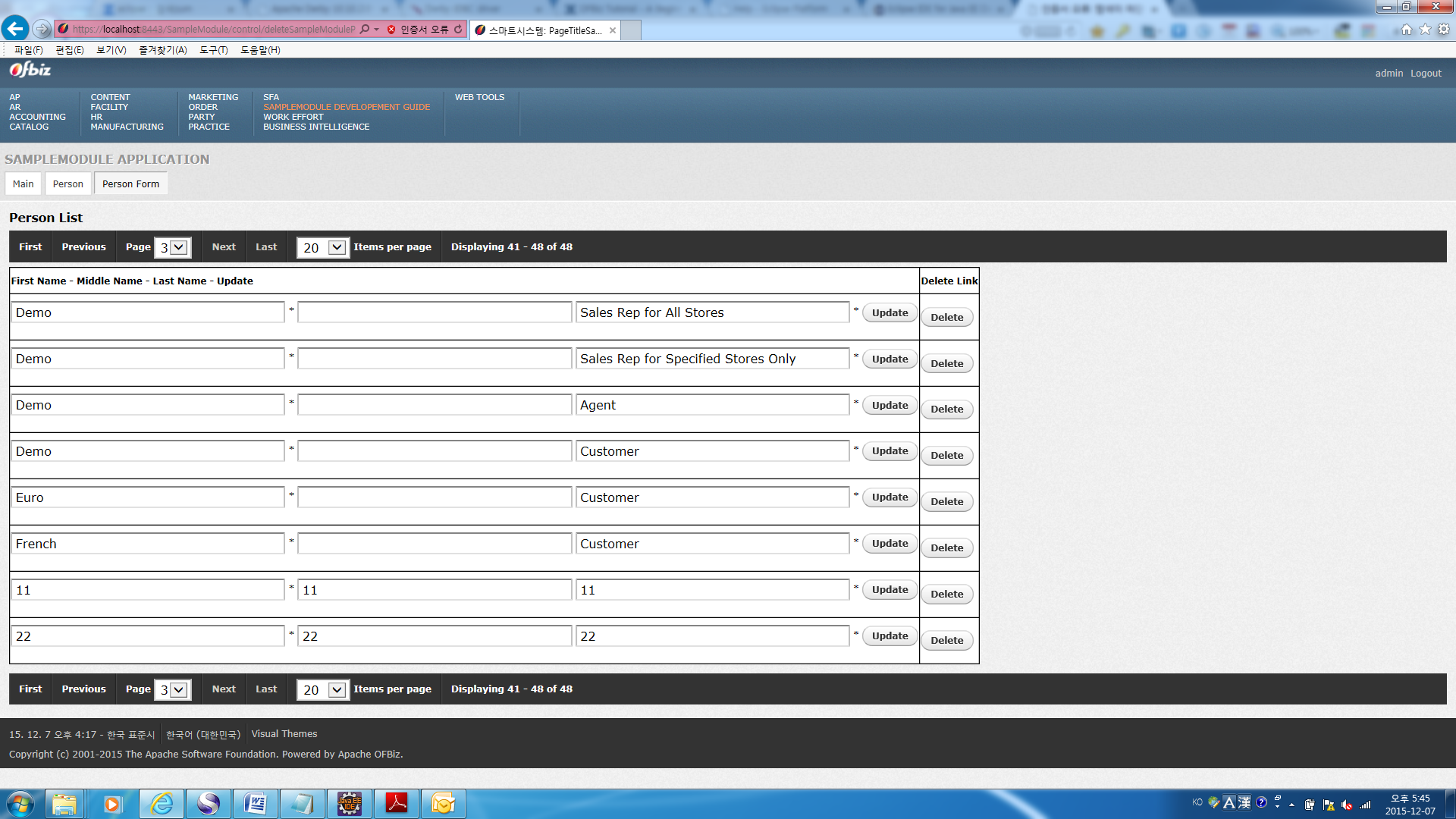
# 편집 가능한 필드로 List form을 변환하여(Ref. ListExampleItems from ExampleForms.xml) Update, Delete옵션도 추가하고 같은 화면에서 정보 수정이 가능한 폼도 위치하게 하는데 자세한 내용은 아래와 같음:

|  |
| --- |
| <form name=*"ListPersons"* type=*"list"* list-name=*"persons"* list-entry-name=*"person"* target=*"updateSampleModulePerson"*>  <auto-fields-service service-name=*"updateSampleModulePerson"* default-field-type=*"edit"* map-name=*"person"*/>  <field name=*"partyId"*><hidden/></field>  <field name=*"deleteLink"* title=*""* widget-style=*"buttontext"*>  <hyperlink target=*"deleteSampleModulePerson"* description=*"${uiLabelMap.CommonDelete}"* also-hidden=*"false"*>  <parameter param-name=*"partyId"* from-field=*"person.partyId"*/>  </hyperlink>  </field>  <field name=*"submitButton"* title=*"${uiLabelMap.CommonUpdate}"*><submit button-type=*"button"*/></field>  </form> |

1. Controller.xml에 이상과 같이 작성된 폼을 부르기 위한 이들 서비스들의 요청,응답 작성

|  |
| --- |
| <request-map uri=*"createSampleModulePerson"*>  <security https=*"true"* auth=*"true"*/>  <event type=*"service"* invoke=*"createSampleModulePerson"*/>  <response name=*"success"* type=*"view"* value=*"personForm"*/>  </request-map>  <request-map uri=*"updateSampleModulePerson"*>  <security https=*"true"* auth=*"true"*/>  <event type=*"service"* invoke=*"updateSampleModulePerson"*/>  <response name=*"success"* type=*"view"* value=*"personForm"*/>  <response name=*"error"* type=*"view"* value=*"personForm"*/>  </request-map>  <request-map uri=*"deleteSampleModulePerson"*>  <security https=*"true"* auth=*"true"*/>  <event type=*"service"* invoke=*"deleteSampleModulePerson"*/>  <response name=*"success"* type=*"view"* value=*"personForm"*/>  <response name=*"error"* type=*"view"* value=*"personForm"*/>  </request-map>  …………………………………..….  <view-map name=*"personForm"* type=*"screen"* page=*"component://SampleModule/widget/SampleModuleScreens.xml#personForm"*/> |
|  |

어플리케이션을 실행해 보고 아래와 같은 화면을 체크해 보기 바람니다.

**Output Screen:**  


## 이벤트 작성

이벤트는 Java와 minilang 둘 다로 작성 할 수 있다. 계속 하려고 하는 다음 단계 개발은 이러한 이벤트들을 작성하는 것이다.

이벤트들은 간단한 Map Processor를 사용하여 타당성체크와 변환을 위해 사용 되어 진다. 간단한 Map Processor인 Mini-Language는 두 가지 주요 타스크인 타당성 체크(validation)와 변환(conversion)을 수행한다. 어떤 특정 Map에서 또다른 특정 Map으로 이동하는 과정 중에 이러한 타스크를 수행한다. 입력 Map에는 보통 스트링 데이터(Strings)를 주로 포함하나 다른 오브젝트 타입들도 포함한다. 예로서 Integer, Long, Float, Double, java.sql.Date, Time, and Timestamp. 등..

보다 자세한 설명을 하기 전에 다음 링크를 통해 중요한 내용을 확인하기 바랍니다.

: [Mini-Language Guide (Version 2 - Deprecated)](https://cwiki.apache.org/confluence/pages/viewpage.action?pageId=6553867), Old link(<http://ofbiz.apache.org/docs/minilang.html>).

이해를 돕게 하기 위하여 실제 구현해 보기 위해 아래와 같은 단계들을 수행해 볼 것입니다.

1. 이를 위해 어플리케이션 메뉴로 "Events"라는 이름의 메뉴 추가

|  |
| --- |
| <menu name=*"SampleModuleAppBar"* default-menu-item-name=*"main"* id=*"app-navigation"* type=*"simple"* title=*"SampleModule Application"*  default-selected-style=*"selected"* selected-menuitem-context-field-name=*"headerItem"*>  <menu-item name=*"main"* title=*"Main"*><link target=*"main"*/></menu-item>  <menu-item name=*"person"* title=*"Person"*><link target=*"person"*/></menu-item>  <menu-item name=*"personForm"* title=*"Person Form"*><link target=*"personForm"*/></menu-item>  <!-- Part3 Writing Events 2015.12.10 추가 -->  <menu-item name=*"events"* title=*"Events"*><link target=*"Events"*/></menu-item>  </menu>  …………………………………..…. |

1. SampleModuleMenus.xml파일에 또 다른 메뉴인 "EventMenu"라는 메뉴를 만들고 거기에 두 개의 메뉴 아이템을 생성하는데 하나의 이름은 "EventMinilang"로 다른 하나는 "EventJava"로 작성하며 각각은 minilang과 java로 작성 된 것을 보여주기 위해 사용될 것이다.
2. 새로운 사원을 생성하면서 두 가지 요청에 대해 타켓 폼을 달리하여 다른 이벤트가 불려지는 것을 보여 주게 될 것이다

|  |
| --- |
| <!-- Part3 Writing Events 2015.12.10 추가 -->  <menu name=*"EventMenu"* default-menu-item-name=*"eventMinilang"* default-selected-style=*"selected"*  type=*"simple"* menu-container-style=*"button-bar button-style-1"* selected-menuitem-context-field-name=*"headerItem"*>  <menu-item name=*"eventMinilang"* title=*"Create Person --- Event MiniLang"*>  <link target=*"CreateSampleModulePersonEventM"*/>  </menu-item>  <menu-item name=*"eventJava"* title=*"Create Person --- Event Java"*>  <link target=*"CreateSampleModulePersonEventJ"*/>  </menu-item>  </menu>  …………………………………..…. |

1. "New Person - Simple Event"와 "New Person - Java Event"라벨을 화면상에 표시하여 각각 폼 화면의 실행 결과를 식별하는 것이 쉽게 할 것이다.

|  |
| --- |
| <request-map uri=*"CreateSampleModulePersonEventM"*>  <security https=*"false"* auth=*"true"*/>  <event type=*"simple"* path=*"component://SampleModule/script/org/hotwax/SampleModule/SampleModuleEvents.xml"* invoke=*"CreateSampleModulePersonEventM"*/>  <response name=*"success"* type=*"view"* value=*"CreateSampleModulePersonEventM"*/>  </request-map> |

1. 이제 화면 폼의 타겟에 이벤트를 설정하고 이 이벤트를 위한 요청을 controller.xml에 추가하고 단순 이벤트의 경우 주목 할 중요한 점은 아래와 같이 작성 된다는 것이다.
2. 여기서 path는 이벤트가 쓰여진 파일이 존재하는 위치로서 SampleModule/script/org/hotwax/SampleModule과 같고 자바 이벤트에 대한 controller.xml파일의 추가 내용은 아래와 같다.

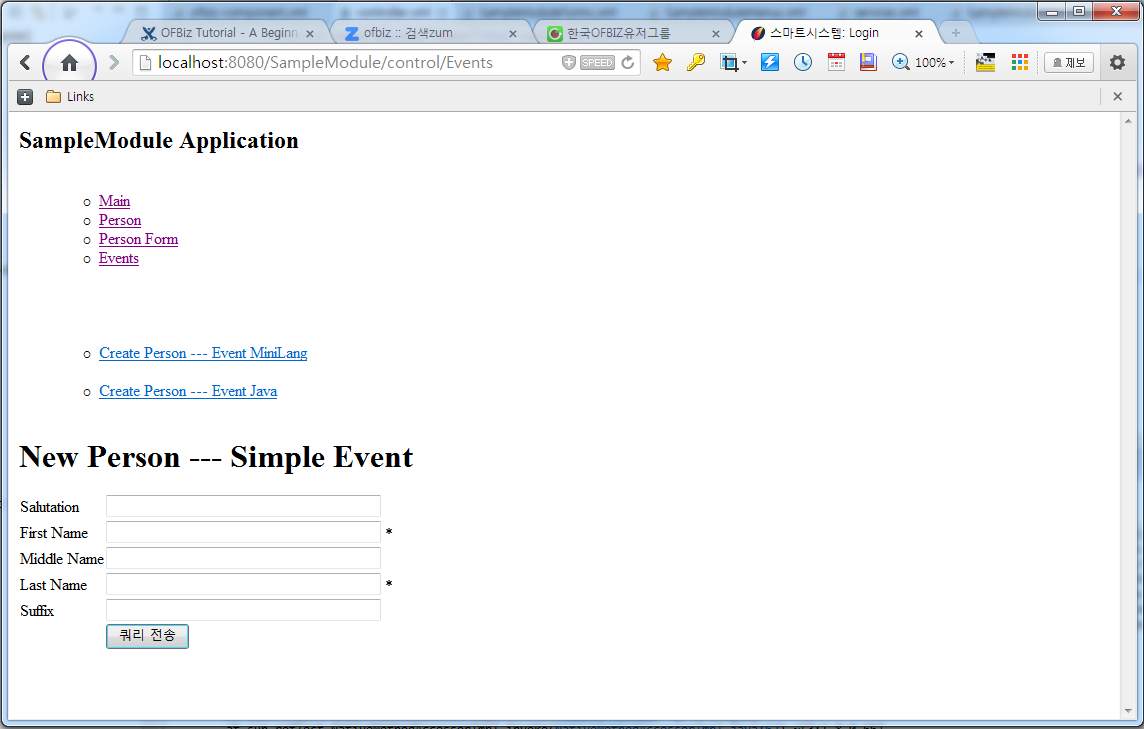
|  |
| --- |
| <request-map uri=*"CreateSampleModulePersonEventJ"*>  <security https=*"false"* auth=*"true"*/>  <event type=*"java"* path=*"org.hotwax.SampleModule.SampleModuleEvents"* invoke=*"createSampleModulePersonJavaEvent"*/>  <response name=*"success"* type=*"view"* value=*"CreateSampleModulePersonEventJ"*/>  </request-map> |

1. Path는 이벤트가 정의된 파일이 위치하는 곳을 가리키는 classpath이고   
   파일이름은 SampleModuleEvents.java가 될것이고 SampleModule/src/org/hotwax/SampleModule.에 작성된다.

## Simple Minilang Event

1. 자 이제 script/org/hotwax/SampleModule/에 SampleModuleEvents.xml.파일을 작성한다
2. SampleModuleEvents.xml.파일에 simple-method엘리먼트의 method-name속성에 CreateSampleModulePersonEventM라는 이름(name)으로 추가한다.   
   추가된 CreateSampleModulePersonEventM은 폼에서 받는 salutation, firstName, middleName, lastName, suffix. 의 5개 필드를 처리하는데 그때 createSampleModulePerson이라는 서비스를 부르게 될 것이고 필드 처리를 위해 이전에 언급했던 단순 Map 프로세서를 사용하게 될 것이다. 이벤트 파일의 내용을 작성하기 위해 아래와 같은 단계를 따른다.:
3. 아래와 같이 폼에서 온 필드를 처리한다:

|  |
| --- |
| <call-map-processor in-map-name=*"parameters"* out-map-name=*"createPersonContext"*>  <simple-map-processor name=*"createPersonMap"*>  <process field=*"firstName"*><copy/><not-empty>  <fail-property property=*"SampleModuleFirstNameMissingError"* resource=*"SampleModuleUiLabels"*/></not-empty>  </process>  <process field=*"middleName"*><copy/></process>  <process field=*"lastName"*><copy/><not-empty>  <fail-property property=*"SampleModuleLastNameIsMissingError"* resource=*"SampleModuleUiLabels"*/></not-empty>  </process>  <process field=*"suffix"*><copy/></process>  </simple-map-processor>  </call-map-processor>  <check-errors/> |

1. 오류시 에러 메시지를 표시하기 위한 SampleModuleFirstNameMissingError 과 같은 UI라벨을 작성.
2. map 에서 서비스로 전달하여 처리 하기 위해 서비스를 호출한다.  
   **Output Screen:**  
   

## Java Event

여기서 작성할 자바 이벤트는 굉장히 단순하다. 작성시 참고하기 위해 어떤 \*Events.java 파일을 참조해도 무방하다..

1. 내용의 주요 부분 시작은 다음과 같이 시작하고:

|  |
| --- |
| public static String createSampleModulePersonJavaEvent(HttpServletRequest request, HttpServletResponse response){      LocalDispatcher dispatcher = (LocalDispatcher) request.getAttribute("dispatcher");      GenericDelegator delegator = (GenericDelegator) request.getAttribute("delegator");  } |

1. 아래와 같이 폼에서 전달되어오는 필드들을 처리 해야 하고

|  |
| --- |
| String salutation = (String) request.getParameter("salutation");  String firstName = (String) request.getParameter("firstName"); |

1. 서비스 "createSampleModulePerson" 에 전달 해야 하는 필드 값들을 위해 아래와 같이 Map을 준비

|  |
| --- |
| Map createPersonCtx = UtilMisc.toMap("salutation", salutation, "firstName", firstName); |

1. 그러고 나서 끝에 "createSampleModulePerson"라는 서비스를 부른다

|  |
| --- |
| try{      Map person = dispatcher.runSync("createSampleModulePerson", createPersonCtx);   }catch (GenericServiceException e){       Debug.logError(e.toString(), module);       return "error";   }  return "success"; |

자바 이벤트를 작성 후에 컴파일 하는 것을 잊지 말기 바란다. Ant를 이용한 컴파일을 위해 컴포넌트 루트 디렉토리에 build.xml파일을 작성하는 것이 필요하다. 예로 hot-deploy/SampleModule/에 "example"컴포넌트 내용을 참조해서 작성하기 바람.   
build.xml 파일에 포함되어야 하는 주요 내용으로 다음과 같고:

|  |
| --- |
| <target name="classpath">      <path id="local.class.path">          <fileset dir="../../framework/base/lib/j2eespecs" includes="\*.jar"/>      </path>  </target> |

이것은 아래와 같은 클래스들을 위해 필요하게 될 것이다

|  |
| --- |
| import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse; |

그래서 build.xml 파일을 작성하고 컴파일 하면 컴포넌트 루트 디렉토리 아래 build라는 디랙토리가 생기고 그안에 \*.jar와 클래스 파일들이 존재하게 된다.   
단순 이밴트(the simple event)를 위해 ofbiz-component.xml 파일에 <classpath type="dir" location="script"/>를 추가하는 것을 잊지 말기 바라고 또 자바 이벤트의 경우 <classpath type="jar" location="build/lib/\*"/>를 추가하는 것을 잊지 말기 바란다.

# Part 4

## ECA(Event Condition Action)

**ECA** : 이것은 3가지의 조합이다: 이벤트, 이벤트에 대한 조건, 이벤트에 대한 액션. 또

It is a combinition of 3 things: an ***event***, ***conditions*** per event, and ***actions*** per event. It is a rule used to trigger an action upon the execution of an event when certain conditions are met. When a service is called for example a lookup is performed to see if any ECAs are defined for this event. Events include before authentication, before IN parameter validation, before actual service invocation, before OUT parameter validation, before transaction commit, or before the service returns. Next, each condition in the ECA definition is evaluated and if all come back as true, each action is performed. An action is just a service which must be defined to work with the parameters already in the service's context. There is no limit to the number of conditions or actions each ECA may define.  
For more details on this visit :  [Service Engine Guide](https://cwiki.apache.org/confluence/display/OFBIZ/Service+Engine+Guide)

1. **SECA (Service Event Condition Action)** : This is used when we want to trigger another service(action) on the execution of a service when certain conditions are met.
2. EECA (Entity Event Condition Action)**: This is used when we want to trigger a service on the creation of a record for an entity when certain conditions are met.  
   For the implementation of ECA again we will be following the same approach for screens, menus by following steps:**
3. Add one more application menu named "ECA" to the practice application's menu bar.(Do the needful entries for target in controller.xml)
4. Now create another menu called "EcaMenu" in the PracticeMenus.xml file. This menu will have two menu items named "seca" and "eeca". For each of these, two screens will be needed that use the "CreatePerson" form which we created above. (in personForm screen)

|  |
| --- |
| <menu name="EcaMenu" default-menu-item-name="seca" default-selected-style="selected"         type="simple" menu-container-style="button-bar button-style-2" selected-menuitem-context-field-name="headerItem">        <menu-item name="seca" title="Create Person --- SECA">            <link target="seca"/>        </menu-item>        <menu-item name="eeca" title="Create Person --- EECA">            <link target="eeca"/>        </menu-item>  </menu> |

## SECA

1. For this you have to write another service by name "createPartyRoleVisitor",  which will be setting the role for the party which will be created by "createPracticePerson" service.  
   The service "createPartyRoleVisitor" will be triggered by the seca rule which you will define for service "createPracticePerson".  
   In the new service involved entity will by "PartyRole". In "createPartyRoleVisitor" just call service "createPartyRole" which is already implemented.
2. Now you have to create a file by name "secas.xml" in "servicedef" directory. Seca definition will come here. (Take reference from secas.xml of "party" component). This will be

|  |
| --- |
| <eca service="createPracticePerson" event="commit">      <condition field-name="partyId" operator="is-not-empty"/>      <action service="createPartyRoleVisitor" mode="sync"/>  </eca> |

1. Do an entry in ofbiz-component.xml file for this seca definition to to be loaded. It will be :  
   <service-resource type="eca" loader="main" location="servicedef/secas.xml"/>  
   Don't forget to restart the server after doing this entry.  
   Now run the service through form and check the records in PartyRole entity. You will find a role is set for the party created because synchrounously you have triggered another service by seca rule for setting up the role for the party created.

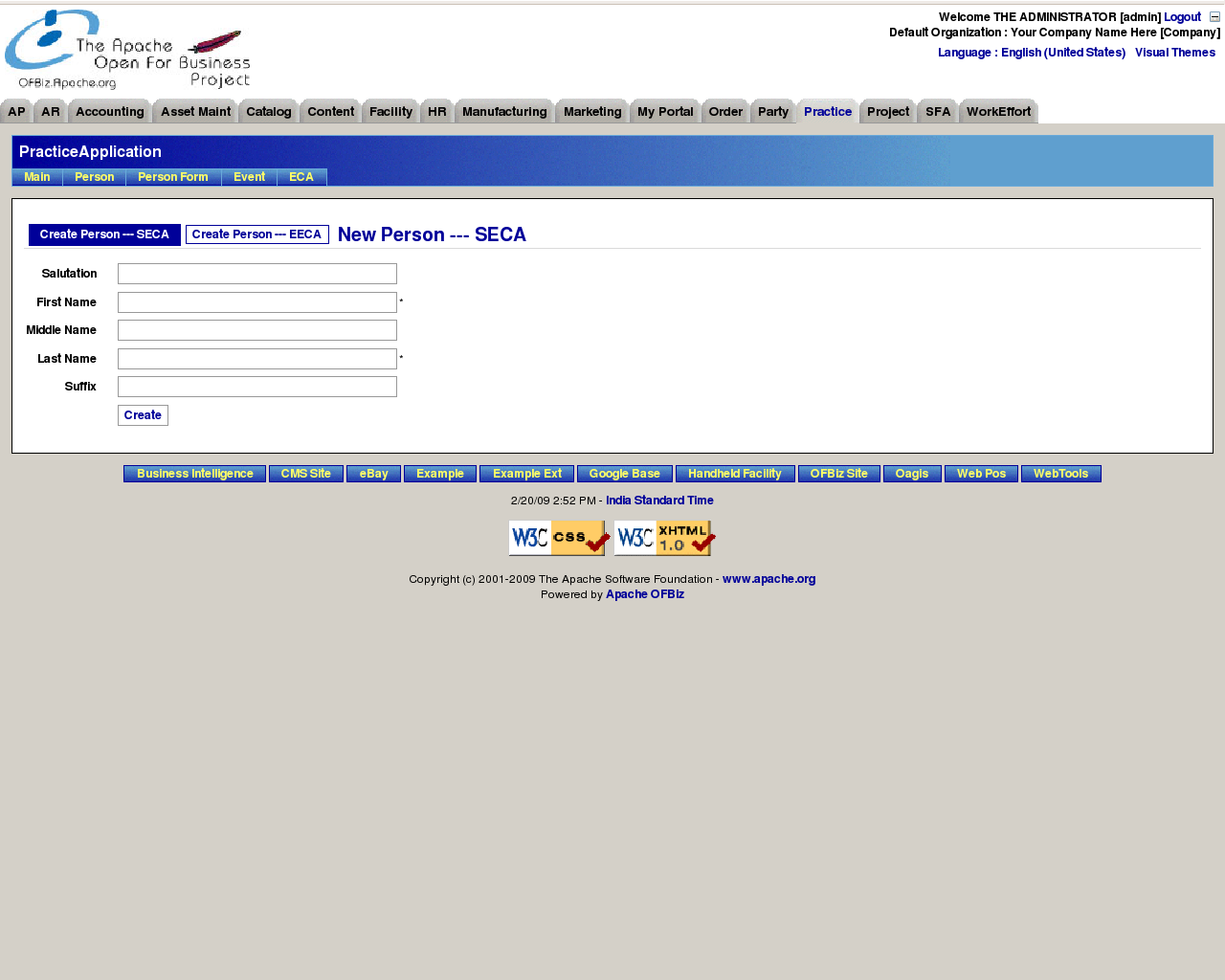
## EECA

1. For this you have to write another service by name "createPartyRoleCustomer",  which will be setting the role for the party which will be created means when a record for the entity "Party" will be created this service will be triggered for setting a role customer for that party. The service "createPartyRoleCustomer" will be similar to "createPartyRoleVisitor".
2. Now you have to create a file by name "eecas.xml" in "entitydef" directory, which will be in your component directory "practice". Eeca definition will come here. (Take reference from eecas.xml of "accounting" component). This will be :

|  |
| --- |
| <!-- To create party role whenever a party is created -->  <eca entity="Party" operation="create" event="return">      <condition field-name="partyId" operator="is-not-empty"/>      <action service="createPartyRoleCustomer" mode="sync"/>  </eca> |

1. Do an entry in ofbiz-component.xml file for this seca definition to to be loaded. It will be :

|  |
| --- |
| <entity-resource type="eca" reader-name="main" loader="main" location="entitydef/eecas.xml"/> |

Don't forget to restart the server after doing this entry.  
Now run the service through form and check the records in PartyRole entity. You will find a role is set for the party created because synchrounously you have triggered a service by eeca rule for setting up the role for the party created.  
The main difference here is that you are triggering a service when an operation is performed on the entity. In our case it is "create".  
**Note** **:** Here you have created a saparate menu to understand the concept separately. As you written seca for the service "createPracticePerson", so where ever in your practice application you will be calling this service that seca will trigger "createPartyRoleVisitor" and on the other hand when a party will be created "createPartyRoleCustomer" will be triggered.  
**Output Screen :**  


## Group Service

    Group services are used to call more then one services as a group. Service groups are a set of services which should run when calling the initial service. You define a service using the group service engine, and include all the parameters/attributes needed for all the services in the group. The location attribute is not needed for groupservices, the invoke attribute defines the name of the group to run. When this service is invoked the group is called and the services defined in the group are called as defined.  
For mor details on this visit :<http://ofbiz.apache.org/docs/services.html>  
For the implementation of Group service follow these steps:

1. Add another menu item to applicatoin menu bar by name "Group Service".(Do the needful entries for target in controller.xml)
2. Now create new screen and a form for creation of the person because the target for the form will be the group service which we will be defining.  
   **Note** **:** Now the time is to define the group service. We will be defining the group service for the services which we have implemented for this practice application.
3. You will be defining the service group in services.xml file.(Take reference from services.xml of party component).  
   Just write one more service which will be setting the role "CLIENT" for the party which will be created by createPracticePerson Service.   
    Create a group service by name "partyGroup" like :

|  |
| --- |
| <!-- Group service -->  <service name="partyGroup" engine="group" auth="true">      <description>Creates a party, person and party role Client</description>      <group>          <invoke name="createPracticePerson" result-to-context="true"/>          <invoke name="createPartyRoleClient"/>      </group>  </service> |

Don't forget to restart the server before testing it.

## Interface

The interface service engine has been implemented to help with defining services which share a number of the same parameters. An interface service cannot be invoked, but rather is a defined service which other services inherit from. Each interface service will be defined using the interface engine.  
For more details on this visit :  <http://ofbiz.apache.org/docs/services.html>  
For implemeting the interface follow these steps:

1. Add another menu item to applicatoin menu bar by name "Interface".(Do the needful entries for target in controller.xml)
2. Create new screen, form and service for creating a person. Here service will implement the interface. (For creating interface take reference from services\_fixedasset.xml of accounting component) it will be like :

|  |
| --- |
| <!-- Peson Interface -->  <service name="createPersonInterface" engine="interface" location="" invoke="">      <attribute name="firstName" mode="IN" type="String" optional="false"/>      <attribute name="middleName" mode="IN" type="String" optional="true"/>      <attribute name="lastName" mode="IN" type="String" optional="false"/>      <attribute name="suffix" mode="IN" type="String" optional="true"/>  </service>  <service name="createPracticePersonInterfaceService" engine="simple"          location="org/hotwax/practice/PracticeServices.xml" invoke="createPracticePersonInterfaceService" auth="false">      <description>Creates a new Person</description>      <implements service="createPersonInterface"/>      <attribute name="partyId" mode="OUT" type="String" optional="false"/>      <override name="suffix" optional="false"/>  </service> |

Here we are implementing an interface and overriding the behaviour of the attribute "suffix", which will have effect when this service will be in action.  
Implementation of service createPracticePersonInterfaceService will be the same as createPracticePerson.  
Don't forget to restart the server after this implementation.